

**inmos®**

# IMS T800 transputer

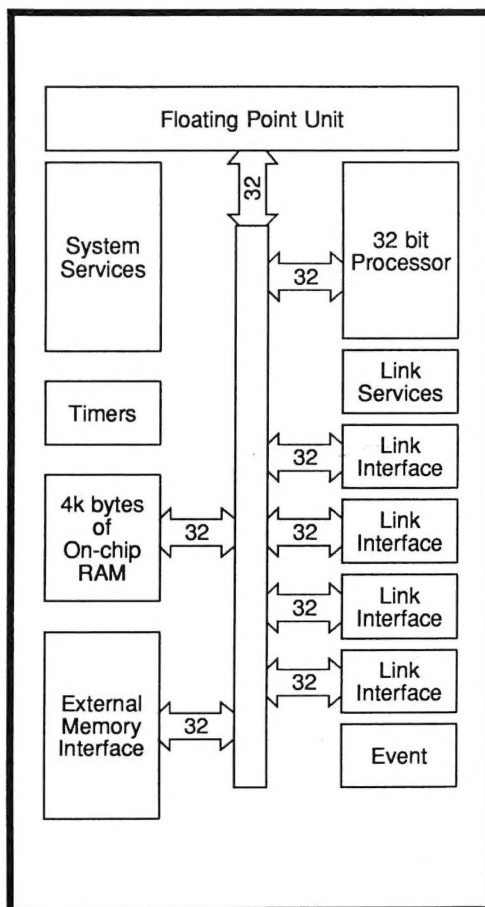
## Engineering Data

### FEATURES

- 32 bit architecture
- 33 ns internal cycle time
- 30 MIPS (peak) instruction rate
- 4.3 Mflops (peak) instruction rate
- 64 bit on-chip floating point unit which conforms to IEEE 754
- 4 Kbytes on-chip static RAM
- 120 Mbytes/sec sustained data rate to internal memory
- 4 Gbytes directly addressable external memory
- 40 Mbytes/sec sustained data rate to external memory
- 630 ns response to interrupts
- Four INMOS serial links 5/10/20 Mbits/sec
- Bi-directional data rate of 2.4 Mbytes/sec per link
- High performance graphics support with block move instructions
- Boot from ROM or communication links
- Single 5 MHz clock input
- Single +5V  $\pm 5\%$  power supply
- MIL-STD-883C processing will be available

### APPLICATIONS

- Scientific and mathematical applications
- High speed multi processor systems
- High performance graphics processing
- Supercomputers
- Workstations and workstation clusters
- Digital signal processing
- Accelerator processors
- Distributed databases
- System simulation
- Telecommunications
- Robotics
- Fault tolerant systems
- Image processing
- Pattern recognition
- Artificial intelligence





## Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Pin designations</b>	<b>5</b>
<b>3</b>	<b>Processor</b>	<b>6</b>
3.1	Registers	6
3.2	Instructions	7
3.2.1	Direct functions	7
3.2.2	Prefix functions	7
3.2.3	Indirect functions	8
3.2.4	Expression evaluation	8
3.2.5	Efficiency of encoding	8
3.3	Processes and concurrency	9
3.4	Priority	10
3.5	Communications	10
3.6	Timers	11
<b>4</b>	<b>Instruction set summary</b>	<b>12</b>
4.1	Descheduling points	13
4.2	Error instructions	14
4.3	Floating point errors	14
<b>5</b>	<b>Floating point unit</b>	<b>21</b>
<b>6</b>	<b>System services</b>	<b>23</b>
6.1	Power	23
6.2	CapPlus, CapMinus	23
6.3	ClockIn	23
6.4	ProcSpeedSelect0-2	24
6.5	Reset	25
6.6	Bootstrap	25
6.7	Peek and poke	27
6.8	Analyse	27
6.9	Error, ErrorIn	28
<b>7</b>	<b>Memory</b>	<b>29</b>
<b>8</b>	<b>External memory interface</b>	<b>31</b>
8.1	ProcClockOut	31
8.2	Tstates	31
8.3	Internal access	32
8.4	MemAD2-31	33
8.5	MemnotWrD0	33
8.6	MemnotRfD1	33
8.7	notMemRd	33
8.8	notMemS0-4	33
8.9	notMemWrB0-3	37
8.10	MemConfig	40
8.10.1	Internal configuration	40
8.10.2	External configuration	41
8.11	notMemRf	47

<b>8.12</b>	<b>MemWait</b>	<b>48</b>
<b>8.13</b>	<b>MemReq, MemGranted</b>	<b>50</b>
<b>9</b>	<b>Events</b>	<b>52</b>
<b>10</b>	<b>Links</b>	<b>53</b>
<b>11</b>	<b>Electrical specifications</b>	<b>56</b>
<b>11.1</b>	<b>DC electrical characteristics</b>	<b>56</b>
<b>11.2</b>	<b>Equivalent circuits</b>	<b>57</b>
<b>11.3</b>	<b>AC timing characteristics</b>	<b>58</b>
<b>11.4</b>	<b>Power rating</b>	<b>59</b>
<b>12</b>	<b>Performance</b>	<b>61</b>
<b>12.1</b>	<b>Performance overview</b>	<b>61</b>
<b>12.2</b>	<b>Fast multiply, TIMES</b>	<b>63</b>
<b>12.3</b>	<b>Arithmetic</b>	<b>64</b>
<b>12.4</b>	<b>IMS T800 floating point operations</b>	<b>65</b>
	<b>12.4.1 IMS T800 floating point functions</b>	<b>65</b>
	<b>12.4.2 IMS T800 special purpose functions and procedures</b>	<b>66</b>
<b>12.5</b>	<b>Effect of external memory</b>	<b>66</b>
<b>12.6</b>	<b>Interrupt latency</b>	<b>67</b>
<b>13</b>	<b>Package specifications</b>	<b>68</b>
<b>13.1</b>	<b>84 pin grid array package</b>	<b>68</b>
<b>14</b>	<b>Ordering</b>	<b>70</b>

## 1 Introduction

The IMS T800 transputer is a 32 bit CMOS microcomputer with a 64 bit floating point unit and graphics support. It has 4 Kbytes on-chip RAM for high speed processing, a configurable memory interface and four standard INMOS communication links. The instruction set achieves efficient implementation of high level languages and provides direct support for the OCCAM model of concurrency when using either a single transputer or a network. Procedure calls, process switching and typical interrupt latency are sub-microsecond.

For convenience of description, the IMS T800 operation is split into the basic blocks shown in figure 1.1.

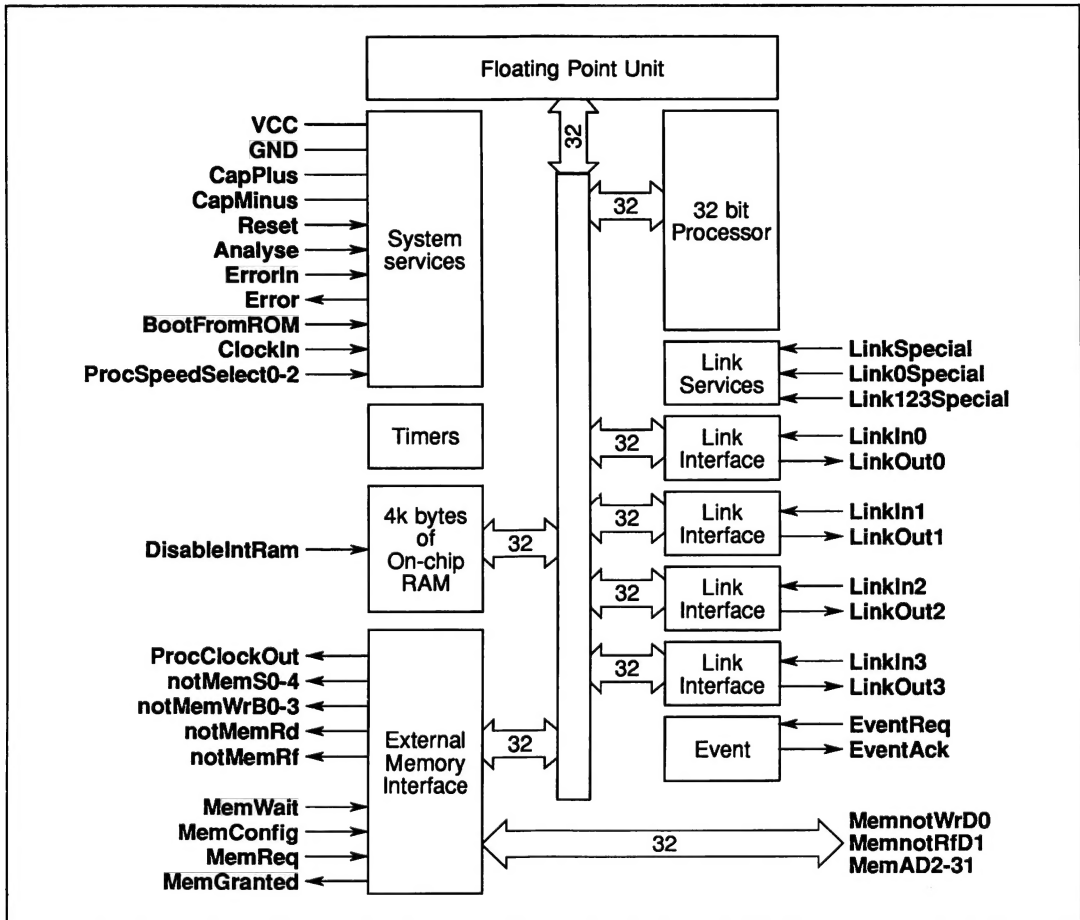


Figure 1.1 IMS T800 block diagram

The processor speed of a device can be pin-selected in stages from 17.5 MHz up to the maximum allowed for the part. A device running at 30 MHz achieves an instruction throughput of 30 MIPS peak and 15 MIPS sustained. The extended temperature version of the device complies with MIL-STD-883C.

The IMS T800 provides high performance arithmetic and floating point operations. The 64 bit floating point unit provides single and double length operation to the ANSI-IEEE 754-1985 standard for floating point arithmetic. It is able to perform floating point operations concurrently with the processor, sustaining a rate of 1.5 Mflops at a processor speed of 20 MHz and 2.25 Mflops at 30 MHz.

High performance graphics support is provided by microcoded block move instructions which operate at the speed of memory. The two-dimensional block move instructions provide for contiguous block moves as well as block copying of either non-zero bytes of data only or zero bytes only. Block move instructions can be used to provide graphics operations such as text manipulation, windowing, panning, scrolling and screen updating.

Cyclic redundancy checking (CRC) instructions are available for use on arbitrary length serial data streams, to provide error detection where data integrity is critical. Another feature of the IMS T800, useful for pattern recognition, is the facility to count bits set in a word.

The IMS T800 can directly access a linear address space of 4 Gbytes. The 32 bit wide memory interface uses multiplexed data and address lines and provides a data rate of up to 4 bytes every 100 nanoseconds (40 Mbytes/sec) for a 30 MHz device. A configurable memory controller provides all timing, control and DRAM refresh signals for a wide variety of mixed memory systems.

System Services include processor reset and bootstrap control, together with facilities for error analysis. Error signals may be daisy-chained in multi-transputer systems.

The standard INMOS communication links allow networks of transputer family products to be constructed by direct point to point connections with no external logic. The IMS T800 links support the standard operating speed of 10 Mbits/sec, but also operate at 5 or 20 Mbits/sec. Each link can transfer data bi-directionally at up to 2.35 Mbytes/sec.

The transputer is designed to implement the OCCAM language, detailed in the OCCAM Reference Manual, but also efficiently supports other languages such as C, Pascal and Fortran. Access to the transputer at machine level is seldom required, but if necessary refer to The Transputer Instruction Set - A Compiler Writers' Guide.

This data sheet supplies hardware implementation and characterisation details for the IMS T800. It is intended to be read in conjunction with the Transputer Architecture section of the Transputer Databook, which details the architecture of the transputer and gives an overview of OCCAM.

## 2 Pin designations

Table 2.1 IMS T800 system services

Pin	In/Out	Function
<b>VCC, GND</b>		Power supply and return
<b>CapPlus, CapMinus</b>		External capacitor for internal clock power supply
<b>ClockIn</b>	in	Input clock
<b>ProcSpeedSelect0-2</b>	in	Processor speed selectors
<b>Reset</b>	in	System reset
<b>Error</b>	out	Error indicator
<b>ErrorIn</b>	in	Error daisychain input
<b>Analyse</b>	in	Error analysis
<b>BootFromRom</b>	in	Boot from external ROM or from link
<b>DisableIntRAM</b>	in	Disable internal RAM
<b>DoNotWire</b>		Must not be wired

Table 2.2 IMS T800 external memory interface

Pin	In/Out	Function
<b>ProcClockOut</b>	out	Processor clock
<b>MemnotWrD0</b>	in/out	Multiplexed data bit 0 and write cycle warning
<b>MemnotRfD1</b>	in/out	Multiplexed data bit 1 and refresh warning
<b>MemAD2-31</b>	in/out	Multiplexed data and address bus
<b>notMemRd</b>	out	Read strobe
<b>notMemWrB0-3</b>	out	Four byte-addressing write strobes
<b>notMemS0-4</b>	out	Five general purpose strobes
<b>notMemRf</b>	out	Dynamic memory refresh indicator
<b>MemWait</b>	in	Memory cycle extender
<b>MemReq</b>	in	Direct memory access request
<b>MemGranted</b>	out	Direct memory access granted
<b>MemConfig</b>	in	Memory configuration data input

Table 2.3 IMS T800 event

Pin	In/Out	Function
<b>EventReq</b>	in	Event request
<b>EventAck</b>	out	Event request acknowledge

Table 2.4 IMS T800 link

Pin	In/Out	Function
<b>LinkIn0-3</b>	in	Four serial data input channels
<b>LinkOut0-3</b>	out	Four serial data output channels
<b>LinkSpecial</b>	in	Select non-standard speed as 5 or 20 Mbits/sec
<b>Link0Special</b>	in	Select special speed for Link 0
<b>Link123Special</b>	in	Select special speed for Links 1,2,3

Signal names are prefixed by **not** if they are active low, otherwise they are active high.  
Pinout details for various packages are given on page 68.

### 3 Processor

The 32 bit processor contains instruction processing logic, instruction and work pointers, and an operand register. It directly accesses the high speed 4 Kbyte on-chip memory, which can store data or program. Where larger amounts of memory or programs in ROM are required, the processor has access to 4 Gbytes of memory via the External Memory Interface (EMI).

#### 3.1 Registers

The design of the transputer processor exploits the availability of fast on-chip memory by having only a small number of registers; six registers are used in the execution of a sequential process. The small number of registers, together with the simplicity of the instruction set, enables the processor to have relatively simple (and fast) data-paths and control logic. The six registers are:

The workspace pointer which points to an area of store where local variables are kept.

The instruction pointer which points to the next instruction to be executed.

The operand register which is used in the formation of instruction operands.

The *A*, *B* and *C* registers which form an evaluation stack.

*A*, *B* and *C* are sources and destinations for most arithmetic and logical operations. Loading a value into the stack pushes *B* into *C*, and *A* into *B*, before loading *A*. Storing a value from *A*, pops *B* into *A* and *C* into *B*.

Expressions are evaluated on the evaluation stack, and instructions refer to the stack implicitly. For example, the *add* instruction adds the top two values in the stack and places the result on the top of the stack. The use of a stack removes the need for instructions to respecify the location of their operands. Statistics gathered from a large number of programs show that three registers provide an effective balance between code compactness and implementation complexity.

No hardware mechanism is provided to detect that more than three values have been loaded onto the stack. It is easy for the compiler to ensure that this never happens.

Any location in memory can be accessed relative to the workspace pointer, enabling the workspace to be of any size.

Further register details are given in The Transputer Instruction Set - A Compiler Writers' Guide.

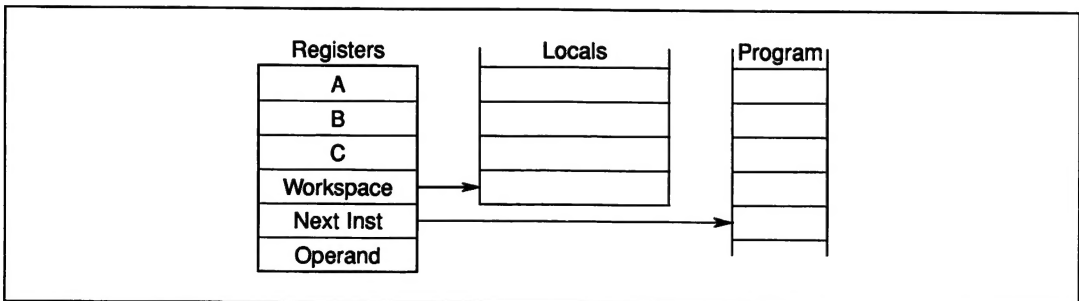


Figure 3.1 Registers



### 3.2 Instructions

The instruction set has been designed for simple and efficient compilation of high-level languages. All instructions have the same format, designed to give a compact representation of the operations occurring most frequently in programs.

Each instruction consists of a single byte divided into two 4-bit parts. The four most significant bits of the byte are a function code and the four least significant bits are a data value.

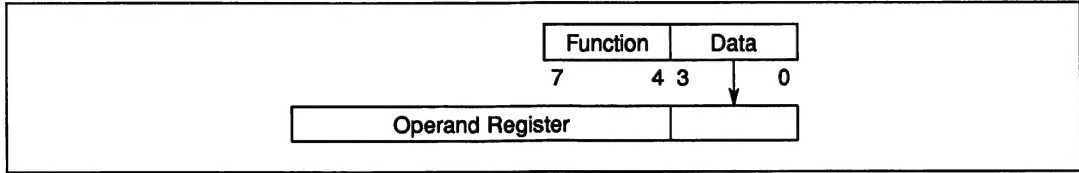


Figure 3.2 Instruction format

#### 3.2.1 Direct functions

The representation provides for sixteen functions, each with a data value ranging from 0 to 15. Ten of these, shown in table 3.1, are used to encode the most important functions.

Table 3.1 Direct functions

<i>load constant</i>	<i>add constant</i>	
<i>load local</i>	<i>store local</i>	<i>load local pointer</i>
<i>load non-local</i>	<i>store non-local</i>	
<i>jump</i>	<i>conditional jump</i>	<i>call</i>

The most common operations in a program are the loading of small literal values and the loading and storing of one of a small number of variables. The *load constant* instruction enables values between 0 and 15 to be loaded with a single byte instruction. The *load local* and *store local* instructions access locations in memory relative to the workspace pointer. The first 16 locations can be accessed using a single byte instruction.

The *load non-local* and *store non-local* instructions behave similarly, except that they access locations in memory relative to the A register. Compact sequences of these instructions allow efficient access to data structures, and provide for simple implementations of the static links or displays used in the implementation of high level programming languages such as OCCAM, C, Fortran, Pascal or ADA.

#### 3.2.2 Prefix functions

Two more function codes allow the operand of any instruction to be extended in length; *prefix* and *negative prefix*.

All instructions are executed by loading the four data bits into the least significant four bits of the operand register, which is then used as the instruction's operand. All instructions except the prefix instructions end by clearing the operand register, ready for the next instruction.

The *prefix* instruction loads its four data bits into the operand register and then shifts the operand register up four places. The *negative prefix* instruction is similar, except that it complements the operand register before shifting it up. Consequently operands can be extended to any length up to the length of the operand register by a sequence of prefix instructions. In particular, operands in the range -256 to 255 can be represented using one prefix instruction.

The use of prefix instructions has certain beneficial consequences. Firstly, they are decoded and executed in the same way as every other instruction, which simplifies and speeds instruction decoding. Secondly, they simplify language compilation by providing a completely uniform way of allowing any instruction to take an operand of any size. Thirdly, they allow operands to be represented in a form independent of the processor wordlength.

### 3.2.3 Indirect functions

The remaining function code, *operate*, causes its operand to be interpreted as an operation on the values held in the evaluation stack. This allows up to 16 such operations to be encoded in a single byte instruction. However, the prefix instructions can be used to extend the operand of an *operate* instruction just like any other. The instruction representation therefore provides for an indefinite number of operations.

Encoding of the indirect functions is chosen so that the most frequently occurring operations are represented without the use of a prefix instruction. These include arithmetic, logical and comparison operations such as *add*, *exclusive or* and *greater than*. Less frequently occurring operations have encodings which require a single prefix operation.

### 3.2.4 Expression evaluation

Evaluation of expressions sometimes requires use of temporary variables in the workspace, but the number of these can be minimised by careful choice of the evaluation order.

Table 3.2 Expression evaluation

Program	Mnemonic
$x := 0$	<i>ldc</i> 0
	<i>stl</i> x
$x := \#24$	<i>pfix</i> 2
	<i>ldc</i> 4
	<i>stl</i> x
$x := y + z$	<i>ldl</i> y
	<i>ldl</i> z
	<i>add</i>
	<i>stl</i> x

### 3.2.5 Efficiency of encoding

Measurements show that about 70% of executed instructions are encoded in a single byte; that is, without the use of prefix instructions. Many of these instructions, such as *load constant* and *add* require just one processor cycle.

The instruction representation gives a more compact representation of high level language programs than more conventional instruction sets. Since a program requires less store to represent it, less of the memory bandwidth is taken up with fetching instructions. Furthermore, as memory is word accessed the processor will receive four instructions for every fetch.

Short instructions also improve the effectiveness of instruction pre-fetch, which in turn improves processor performance. There is an extra word of pre-fetch buffer, so the processor rarely has to wait for an instruction fetch before proceeding. Since the buffer is short, there is little time penalty when a jump instruction causes the buffer contents to be discarded.

### 3.3 Processes and concurrency

A process starts, performs a number of actions, and then either stops without completing or terminates complete. Typically, a process is a sequence of instructions. A transputer can run several processes in parallel (concurrently). Processes may be assigned either high or low priority, and there may be any number of each (page 10).

The processor has a microcoded scheduler which enables any number of concurrent processes to be executed together, sharing the processor time. This removes the need for a software kernel.

At any time, a concurrent process may be

- Active*
  - Being executed.
  - On a list waiting to be executed.
- Inactive*
  - Ready to input.
  - Ready to output.
  - Waiting until a specified time.

The scheduler operates in such a way that inactive processes do not consume any processor time. It allocates a portion of the processor's time to each process in turn. Active processes waiting to be executed are held in two linked lists of process workspaces, one of high priority processes and one of low priority processes (page 10). Each list is implemented using two registers, one of which points to the first process in the list, the other to the last. In the Linked Process List figure 3.3, process *S* is executing and *P*, *Q* and *R* are active, awaiting execution. Only the low priority process queue registers are shown; the high priority process ones perform in a similar manner.

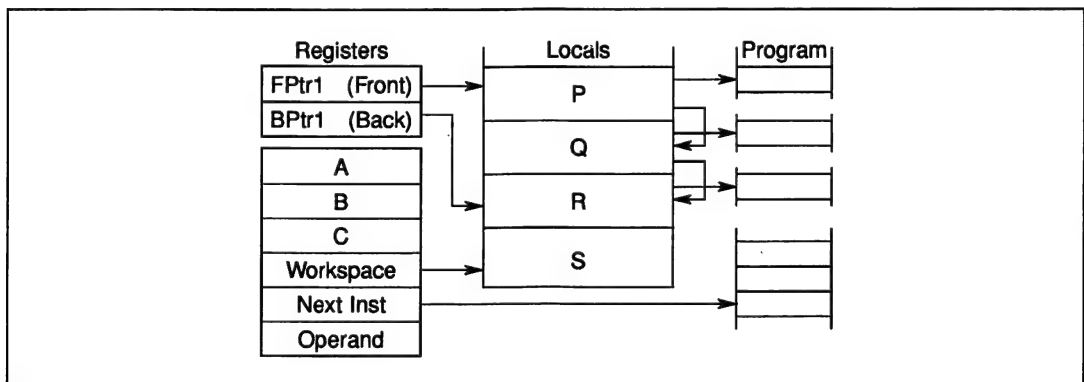


Figure 3.3 Linked process list

Table 3.3 Priority queue control registers

Function	High Priority	Low Priority
Pointer to front of active process list	<i>Fptr0</i>	<i>Fptr1</i>
Pointer to back of active process list	<i>Bptr0</i>	<i>Bptr1</i>

Each process runs until it has completed its action, but is descheduled whilst waiting for communication from another process or transputer, or for a time delay to complete. In order for several processes to operate in parallel, a low priority process is only permitted to run for a maximum of two time slices before it is forcibly descheduled at the next descheduling point (page 13). The time slice period is 5120 cycles of the external 5 MHz clock, giving ticks approximately 1 ms apart.

A process can only be descheduled on certain instructions, known as descheduling points (page 13). As a result, an expression evaluation can be guaranteed to execute without the process being timesliced part way through.

Whenever a process is unable to proceed, its instruction pointer is saved in the process workspace and the next process taken from the list. Process scheduling pointers are updated by instructions which cause scheduling operations, and should not be altered directly. Actual process switch times are less than 1  $\mu$ s, as little state needs to be saved and it is not necessary to save the evaluation stack on rescheduling.

The processor provides a number of special operations to support the process model, including *start process* and *end process*. When a main process executes a parallel construct, *start process* instructions are used to create the necessary additional concurrent processes. A *start process* instruction creates a new process by adding a new workspace to the end of the scheduling list, enabling the new concurrent process to be executed together with the ones already being executed. When a process is made active it is always added to the end of the list, and thus cannot pre-empt processes already on the same list.

The correct termination of a parallel construct is assured by use of the *end process* instruction. This uses a workspace location as a counter of the parallel construct components which have still to terminate. The counter is initialised to the number of components before the processes are *started*. Each component ends with an *end process* instruction which decrements and tests the counter. For all but the last component, the counter is non zero and the component is descheduled. For the last component, the counter is zero and the main process continues.

### 3.4 Priority

The IMS T800 supports two levels of priority. Priority 1 (low priority) processes are executed whenever there are no active priority 0 (high priority) processes.

High priority processes are expected to execute for a short time. If one or more high priority processes are able to proceed, then one is selected and runs until it has to wait for a communication, a timer input, or until it completes processing.

If no process at high priority is able to proceed, but one or more processes at low priority are able to proceed, then one is selected.

Low priority processes are periodically timesliced to provide an even distribution of processor time between computationally intensive tasks.

If there are  $n$  low priority processes, then the maximum latency from the time at which a low priority process becomes active to the time when it starts processing is  $2n-2$  timeslice periods. It is then able to execute for between one and two timeslice periods, less any time taken by high priority processes. This assumes that no process monopolises the transputer's time; i.e. it has a distribution of descheduling points (page 13).

Each timeslice period lasts for 5120 cycles of the external 5 MHz input clock (approximately 1 ms at the standard frequency of 5 MHz).

If a high priority process is waiting for an external channel to become ready, and if no other high priority process is active, then the interrupt latency (from when the channel becomes ready to when the process starts executing) is typically 19 processor cycles, a maximum of 78 cycles (assuming use of on-chip RAM). If the floating point unit is not being used at the time then the maximum interrupt latency is only 58 cycles. To ensure this latency, certain instructions are interruptable.

### 3.5 Communications

Communication between processes is achieved by means of channels. Process communication is point-to-point, synchronised and unbuffered. As a result, a channel needs no process queue, no message queue and no message buffer.

A channel between two processes executing on the same transputer is implemented by a single word in memory; a channel between processes executing on different transputers is implemented by point-to-point links. The processor provides a number of operations to support message passing, the most important being *input message* and *output message*.

The *input message* and *output message* instructions use the address of the channel to determine whether the channel is internal or external. Thus the same instruction sequence can be used for both, allowing a process to be written and compiled without knowledge of where its channels are connected.

The process which first becomes ready must wait until the second one is also ready. A process performs an input or output by loading the evaluation stack with a pointer to a message, the address of a channel, and a count of the number of bytes to be transferred, and then executing an *input message* or *output message* instruction. Data is transferred if the other process is ready. If the channel is not ready or is an external one the process will deschedule.

### 3.6 Timers

The transputer has two 32 bit timer clocks which 'tick' periodically. The timers provide accurate process timing, allowing processes to deschedule themselves until a specific time.

One timer is accessible only to high priority processes and is incremented every microsecond, cycling completely in approximately 4295 seconds. The other is accessible only to low priority processes and is incremented every 64 microseconds, giving exactly 15625 ticks in one second. It has a full period of approximately 76 hours.

Table 3.4 Timer registers

<i>Clock0</i>	Current value of high priority (level 0) process clock
<i>Clock1</i>	Current value of low priority (level 1) process clock
<i>TNextReg0</i>	Indicates time of earliest event on high priority (level 0) timer queue
<i>TNextReg1</i>	Indicates time of earliest event on low priority (level 1) timer queue

The current value of the processor clock can be read by executing a *load timer* instruction. A process can arrange to perform a *timer input*, in which case it will become ready to execute after a specified time has been reached. The *timer input* instruction requires a time to be specified. If this time is in the 'past' then the instruction has no effect. If the time is in the 'future' then the process is descheduled. When the specified time is reached the process is scheduled again.

Figure 3.4 shows two processes waiting on the timer queue, one waiting for time 21, the other for time 31.

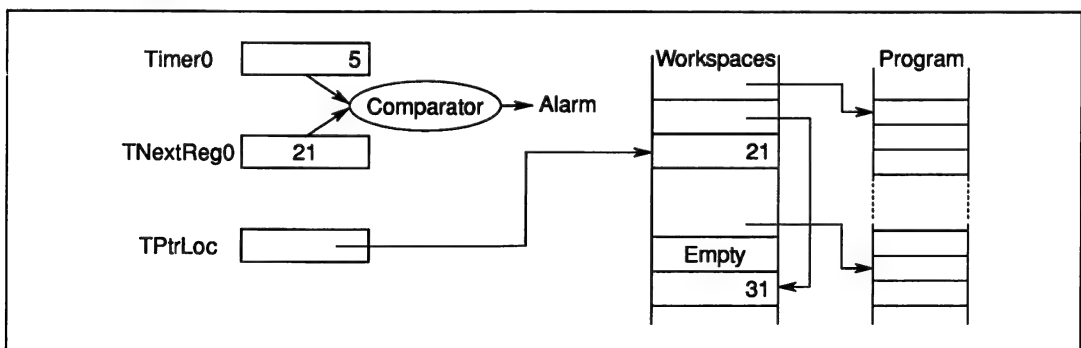


Figure 3.4 Timer registers

## 4 Instruction set summary

The Function Codes table 4.8. gives the basic function code set (page 7). Where the operand is less than 16, a single byte encodes the complete instruction. If the operand is greater than 15, one prefix instruction (*prefix*) is required for each additional four bits of the operand. If the operand is negative the first prefix instruction will be *nfix*.

Table 4.1 *prefix* coding

Mnemonic		Function code	Memory code
<i>ldc</i>	#3	#4	#43
<i>ldc</i>	#35		
<b>is coded as</b>			
<i>prefix</i>	#3	#2	#23
<i>ldc</i>	#5	#4	#45
<i>ldc</i>	#987		
<b>is coded as</b>			
<i>prefix</i>	#9	#2	#29
<i>prefix</i>	#8	#2	#28
<i>ldc</i>	#7	#4	#47
<i>ldc</i>	-31 ( <i>ldc</i> #FFFFFFE1)		
<b>is coded as</b>			
<i>nfix</i>	#1	#6	#61
<i>ldc</i>	#1	#4	#41

Tables 4.9 to 4.27 give details of the operation codes. Where an operation code is less than 16 (e.g. *add*: operation code 05), the operation can be stored as a single byte comprising the *operate* function code F and the operand (5 in the example). Where an operation code is greater than 15 (e.g. *ladd*: operation code 16), the *prefix* function code 2 is used to extend the instruction.

Table 4.2 *operate* coding

Mnemonic		Function code	Memory code
<i>add</i>	(op. code #5)		#F5
<b>is coded as</b>			
<i>opr</i>	<i>add</i>	#F	#F5
<i>ladd</i>	(op. code #16)		#21F6
<b>is coded as</b>			
<i>prefix</i>	#1	#2	#21
<i>opr</i>	#6	#F	#F6

In the Floating Point Operation Codes tables 4.21 to 4.27, a selector sequence code (page 21) is indicated in the Memory Code column by **s**. The code given in the Operation Code column is the indirection code, the operand for the *ldc* instruction.

The FPU and processor operate concurrently, so the actual throughput of floating point instructions is better than that implied by simply adding up the instruction times. For full details see The Transputer Instruction Set - A Compiler Writers' Guide.

The Processor Cycles column refers to the number of periods **TPCLPCL** taken by an instruction executing in internal memory. The number of cycles is given for the basic operation only; where the memory code for an instruction is two bytes, the time for the *prefix* function (one cycle) should be added. For a 20 MHz transputer one cycle is 50 ns. Some instruction times vary. Where a letter is included in the cycles column it is interpreted from table 4.3.

Table 4.3 Instruction set interpretation

Ident	Interpretation
<b>b</b>	Bit number of the highest bit set in register <i>A</i> . Bit 0 is the least significant bit.
<b>m</b>	Bit number of the highest bit set in the absolute value of register <i>A</i> . Bit 0 is the least significant bit.
<b>n</b>	Number of places shifted.
<b>w</b>	Number of words in the message. Part words are counted as full words. If the message is not word aligned the number of words is increased to include the part words at either end of the message.
<b>p</b>	Number of words per row.
<b>r</b>	Number of rows.

The **DE** column of the tables indicates the descheduling/error features of an instruction as described in table 4.4.

Table 4.4 Instruction features

Ident	Feature	See page:
<b>D</b>	The instruction is a descheduling point	13
<b>E</b>	The instruction will affect the <i>Error</i> flag	14, 28
<b>F</b>	The instruction will affect the <i>FP_Error</i> flag	21, 14

#### 4.1 Descheduling points

The instructions in table 4.5 are the only ones at which a process may be descheduled (page 9). They are also the ones at which the processor will halt if the **Analyse** pin is asserted (page 27).

Table 4.5 Descheduling point instructions

<i>input message</i>	<i>output message</i>	<i>output byte</i>	<i>output word</i>
<i>timer alt wait</i>	<i>timer input</i>	<i>stop on error</i>	<i>alt wait</i>
<i>jump</i>	<i>loop end</i>	<i>end process</i>	<i>stop process</i>

## 4.2 Error instructions

The instructions in table 4.6 are the only ones which can affect the *Error* flag (page 28) directly. Note, however, that the floating point unit error flag *FP\_Error* is set by certain floating point instructions (page 14), and that *Error* can be set from this flag by *fpcheckerror*.

Table 4.6 Error setting instructions

<i>add</i>	<i>add constant</i>	<i>subtract</i>	
<i>multiply</i>	<i>fractional multiply</i>	<i>divide</i>	<i>remainder</i>
<i>long add</i>	<i>long subtract</i>	<i>long divide</i>	
<i>set error</i>	<i>testerr</i>	<i>fpcheckerror</i>	
<i>check word</i>	<i>check subscript from 0</i>	<i>check single</i>	<i>check count from 1</i>

## 4.3 Floating point errors

The instructions in table 4.7 are the only ones which can affect the floating point error flag *FP\_Error* (page 21). *Error* is set from this flag by *fpcheckerror* if *FP\_Error* is set.

Table 4.7 Floating point error setting instructions

<i>fpadd</i>	<i>fpsub</i>	<i>fpmul</i>	<i>fpdiv</i>
<i>fpdnladds</i>	<i>fpdnladddb</i>	<i>fpdnlmulsn</i>	<i>fpdnlmuldb</i>
<i>fpremfirst</i>	<i>fpusqrtfirst</i>	<i>fpgt</i>	<i>fpeq</i>
<i>fpuseterror</i>	<i>fpuclearerror</i>	<i>fpctesterror</i>	
<i>fpexpincby32</i>	<i>fpexpdecby32</i>	<i>fpumulby2</i>	<i>fpudivby2</i>
<i>fpur32tor64</i>	<i>fpur64tor32</i>	<i>fpucki32</i>	<i>fpucki64</i>
<i>fpstoi32</i>	<i>fpuabs</i>	<i>fpint</i>	



Table 4.8 IMS T800 function codes

Function Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
0	0X	j	3	jump	D
1	1X	ldlp	1	load local pointer	
2	2X	pfix	1	prefix	
3	3X	ldnl	2	load non-local	
4	4X	ldc	1	load constant	
5	5X	ldnlp	1	load non-local pointer	
6	6X	nfix	1	negative prefix	
7	7X	ldl	2	load local	
8	8X	adc	1	add constant	
9	9X	call	7	call	
A	AX	cj	2	conditional jump (not taken)	E
			4	conditional jump (taken)	
B	BX	ajw	1	adjust workspace	
C	CX	eqc	2	equals constant	
D	DX	stl	1	store local	
E	EX	stnl	2	store non-local	
F	FX	opr	-	operate	

Table 4.9 IMS T800 arithmetic/logical operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
46	24F6	and	1	and	
4B	24FB	or	1	or	
33	23F3	xor	1	exclusive or	
32	23F2	not	1	bitwise not	
41	24F1	shl	n+2	shift left	
40	24F0	shr	n+2	shift right	
05	F5	add	1	add	E
0C	FC	sub	1	subtract	E
53	25F3	mul	38	multiply	E
72	27F2	fmul	35	fractional multiply (no rounding)	E
			40	fractional multiply (rounding)	E
2C	22FC	div	39	divide	E
1F	21FF	rem	37	remainder	E
09	F9	gt	2	greater than	
04	F4	diff	1	difference	
52	25F2	sum	1	sum	
08	F8	prod	b+4 m+5	product for positive register A product for negative register A	

Table 4.10 IMS T800 long arithmetic operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
16	21F6	ladd	2	long add	E
38	23F8	lsub	2	long subtract	
37	23F7	lsum	3	long sum	
4F	24FF	ldiff	3	long diff	
31	23F1	lmul	33	long multiply	
1A	21FA	ldiv	35	long divide	
36	23F6	lshl	n+3	long shift left (n<32)	
			n-28	long shift left(n≥32)	
35	23F5	lshr	n+3	long shift right (n<32)	
			n-28	long shift right (n≥32)	
19	21F9	norm	n+5	normalise (n<32)	E
			n-26	normalise (n≥32)	
			3	normalise (n=64)	

Table 4.11 IMS T800 general operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
00	F0	rev	1	reverse	E
3A	23FA	xword	4	extend to word	
56	25F6	cword	5	check word	
1D	21FD	xdble	2	extend to double	
4C	24FC	csngl	3	check single	E
42	24F2	mint	1	minimum integer	
5A	25FA	dup	1	duplicate top of stack	

Table 4.12 IMS T800 2D block move operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
5B	25FB	move2dinit	8	initialise data for 2D block move	
5C	25FC	move2dall	(2p+23)*r	2D block copy	
5D	25FD	move2dnonzero	(2p+23)*r	2D block copy non-zero bytes	
5E	25FE	move2dzero	(2p+23)*r	2D block copy zero bytes	

Table 4.13 IMS T800 CRC and bit operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
74	27F4	crword	35	calculate crc on word	
75	27F5	crbyte	11	calculate crc on byte	
76	27F6	bitcnt	b+2	count bits set in word	
77	27F7	bitrevword	36	reverse bits in word	
78	27F8	bitrevnbits	n+4	reverse bottom n bits in word	

Table 4.14 IMS T800 indexing/array operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
02	F2	bsub	1	byte subscript	
0A	FA	wsb	2	word subscript	
81	28F1	wsbdb	3	form double word subscript	
34	23F4	bcnt	2	byte count	
3F	23FF	wcnt	5	word count	
01	F1	lb	5	load byte	
3B	23FB	sb	4	store byte	
4A	24FA	move	2w+8	move message	

Table 4.15 IMS T800 timer handling operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
22	22F2	ldtimer	2	load timer	D D
2B	22FB	tin	30	timer input (time future)	
			4	timer input (time past)	
4E	24FE	talt	4	timer alt start	
51	25F1	taltwt	15	timer alt wait (time past)	D D
			48	timer alt wait (time future)	
47	24F7	enbt	8	enable timer	
2E	22FE	dist	23	disable timer	

Table 4.16 IMS T800 input/output operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
07	F7	in	2w+19	input message	D
0B	FB	out	2w+19	output message	D
0F	FF	outword	23	output word	D
0E	FE	outbyte	23	output byte	D
43	24F3	alt	2	alt start	D D
44	24F4	altwt	5	alt wait (channel ready)	
			17	alt wait (channel not ready)	
45	24F5	altend	4	alt end	
49	24F9	enbs	3	enable skip	
30	23F0	diss	4	disable skip	
12	21F2	resetch	3	reset channel	
48	24F8	enbc	7	enable channel (ready)	
			5	enable channel (not ready)	
2F	22FF	disc	8	disable channel	

Table 4.17 IMS T800 control operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
20	22F0	ret	5	return	D D
1B	21FB	ldpi	2	load pointer to instruction	
3C	23FC	gajw	2	general adjust workspace	
06	F6	gcall	4	general call	
21	22F1	lend	10	loop end (loop)	
			5	loop end (exit)	

Table 4.18 IMS T800 scheduling operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
0D	FD	startp	12	start process	D
03	F3	endp	13	end process	D
39	23F9	runp	10	run process	
15	21F5	stopp	11	stop process	
1E	21FE	ldpri	1	load current priority	

Table 4.19 IMS T800 error handling operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
13	21F3	csub0	2	check subscript from 0	E
4D	24FD	ccnt1	3	check count from 1	E
29	22F9	testerr	2	test error false and clear (no error)	E D
			3	test error false and clear (error)	
10	21F0	seterr	1	set error	
55	25F5	stoperr	2	stop on error (no error)	
57	25F7	clrhalterr	1	clear halt-on-error	
58	25F8	sethalterr	1	set halt-on-error	
59	25F9	testhalterr	2	test halt-on-error	

Table 4.20 IMS T800 processor initialisation operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
2A	22FA	testpranal	2	test processor analysing	
3E	23FE	saveh	4	save high priority queue registers	
3D	23FD	savel	4	save low priority queue registers	
18	21F8	sthf	1	store high priority front pointer	
50	25F0	sthb	1	store high priority back pointer	
1C	21FC	stlf	1	store low priority front pointer	
17	21F7	stlb	1	store low priority back pointer	
54	25F4	sttimer	1	store timer	

Table 4.21 IMS T800 floating point load/store operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
8E	28FE	fpdnlsl	2	fp load non-local single	F F F F
8A	28FA	fpdnlld	3	fp load non-local double	
86	28F6	fpdnlsl	4	fp load non-local indexed single	
82	28F2	fpdnlldi	6	fp load non-local indexed double	
9F	29FF	fpdzersl	2	load zero single	
A0	2AF0	fpdzersd	2	load zero double	
AA	2AFA	fpdnladdsl	8/11	fp load non local & add single	
A6	2AF6	fpdnladdld	9/12	fp load non local & add double	
AC	2AFC	fpdnlmulsl	13/20	fp load non local & multiply single	
A8	2AF8	fpdnlmulld	21/30	fp load non local & multiply double	
88	28F8	fpstnlsl	2	fp store non-local single	F F F
84	28F4	fpstnlld	3	fp store non-local double	
9E	29FE	fpstnli32	4	store non-local int32	

Processor cycles are shown as **Typical/Maximum** cycles.

Table 4.22 IMS T800 floating point general operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
AB	2AFB	fpentry	1	floating point unit entry	
A4	2AF4	fpv	1	fp reverse	
A3	2AF3	fpdup	1	fp duplicate	

Table 4.23 IMS T800 floating point rounding operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
22	s	fpurn	1	set rounding mode to round nearest	
06	s	fpurz	1	set rounding mode to round zero	
04	s	fpurp	1	set rounding mode to round positive	
05	s	fpurm	1	set rounding mode to round minus	

Table 4.24 IMS T800 floating point error operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
83	28F3	fpchkerror	1	check fp error	E
9C	29FC	fpsterror	2	test fp error false and clear	F
23	s	fpuseterror	1	set fp error	F
9C	s	fpuclearerror	1	clear fp error	F

Table 4.25 IMS T800 floating point comparison operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
94	29F4	fpgt	4/6	fp greater than	F
95	29F5	fpeq	3/5	fp equality	F
92	29F2	fpordered	3/4	fp orderability	
91	29F1	fpnan	2/3	fp NaN	
93	29F3	fpnotfinite	2/2	fp not finite	
0E	s	fpuchki32	3/4	check in range of type int32	F
0F	s	fpuchki64	3/4	check in range of type int64	F

Processor cycles are shown as **Typical/Maximum** cycles.

Table 4.26 IMS T800 floating point conversion operation codes

Operation Code	Memory Code	Mnemonic	Processor Cycles	Name	D E
07	s	fpur32tor64	3/4	real32 to real64	F
08	s	fpur64tor32	6/9	real64 to real32	F
9D	29FD	fpatoi32	7/9	real to int32	F
96	29F6	fpi32tor32	8/10	int32 to real32	
98	29F8	fpi32tor64	8/10	int32 to real64	
9A	29FA	fpb32tor64	8/8	bit32 to real64	
0D	s	fpunoround	2/2	real64 to real32, no round	
A1	2AF1	fpint	5/6	round to floating integer	F

Processor cycles are shown as **Typical/Maximum** cycles.

Table 4.27 IMS T800 floating point arithmetic operation codes

Operation Code	Memory Code	Mnemonic	Processor cycles		Name	D E
			Single	Double		
87	28F7	fpadd	6/9	6/9	fp add	F
89	28F9	fpsub	6/9	6/9	fp subtract	F
8B	28FB	fpmul	11/18	18/27	fp multiply	F
8C	28FC	fpdiv	16/28	31/43	fp divide	F
0B	s	fpuabs	2/2	2/2	fp absolute	F
8F	28FF	fpremfirst	36/46	36/46	fp remainder first step	F
90	29F0	fpremstep	32/36	32/36	fp remainder iteration	
01	s	fpusqrtfirst	27/29	27/29	fp square root first step	F
02	s	fpusqrtstep	42/42	42/42	fp square root step	
03	s	fpusqrtlast	8/9	8/9	fp square root end	
0A	s	fpuexpinc32	6/9	6/9	multiply by $2^{32}$	F
09	s	fpuexpdec32	6/9	6/9	divide by $2^{32}$	F
12	s	fpumulby2	6/9	6/9	multiply by 2.0	F
11	s	fpudivby2	6/9	6/9	divide by 2.0	F

Processor cycles are shown as **Typical/Maximum** cycles.

## 5 Floating point unit

The 64 bit FPU provides single and double length arithmetic to floating point standard ANSI-IEEE 754-1985. It is able to perform floating point arithmetic concurrently with the central processor unit (CPU), sustaining in excess of 2.25 Mflops on a 30 MHz device. All data communication between memory and the FPU occurs under control of the CPU.

The FPU consists of a microcoded computing engine with a three deep floating point evaluation stack for manipulation of floating point numbers. These stack registers are *FA*, *FB* and *FC*, each of which can hold either 32 bit or 64 bit data; an associated flag, set when a floating point value is loaded, indicates which. The stack behaves in a similar manner to the CPU stack (page 6).

As with the CPU stack, the FPU stack is not saved when rescheduling (page 9) occurs. The FPU can be used in both low and high priority processes. When a high priority process interrupts a low priority one the FPU state is saved inside the FPU. The CPU will service the interrupt immediately on completing its current operation. The high priority process will not start, however, before the FPU has completed its current operation.

Points in an instruction stream where data need to be transferred to or from the FPU are called *synchronisation points*. At a synchronisation point the first processing unit to become ready will wait until the other is ready. The data transfer will then occur and both processors will proceed concurrently again. In order to make full use of concurrency, floating point data source and destination addresses can be calculated by the CPU whilst the FPU is performing operations on a previous set of data. Device performance is thus optimised by minimising the CPU and FPU idle times.

The FPU has been designed to operate on both single length (32 bit) and double length (64 bit) floating point numbers, and returns results which fully conform to the ANSI-IEEE 754-1985 floating point arithmetic standard. Denormalised numbers are fully supported in the hardware. All rounding modes defined by the standard are implemented, with the default being round to nearest.

The basic addition, subtraction, multiplication and division operations are performed by single instructions. However, certain less frequently used floating point instructions are selected by a value in register *A* (when allocating registers, this should be taken into account). A *load constant* instruction *ldc* is used to load register *A*; the *floating point entry* instruction *fentry* then uses this value to select the floating point operation. This pair of instructions is termed a *selector sequence*.

Names of operations which use *fentry* begin with *fpu*. A typical usage, returning the absolute value of a floating point number, would be

*ldc fpuabs; fentry;*

Since the indirection code for *fpuabs* is **0B**, it would be encoded as

Table 5.1 *fentry* coding

Mnemonic		Function code	Memory code
<i>ldc</i>	<i>fpuabs</i>	#4	#4B
<i>fentry</i>	(op. code #AB)		#2AFB
<b>is coded as</b>			
<i>prefix</i>	#A	#2	#2A
<i>opr</i>	#B	#F	#FB

The *remainder* and *square root* instructions take considerably longer than other instructions to complete. In order to minimise the interrupt latency period of the transputer they are split up to form instruction sequences. As an example, the instruction sequence for a single length square root is

*fpusqrtfirst; fpusqrtstep; fpusqrtstep; fpusqrtlast;*

The FPU has its own error flag *FP\_Error*. This reflects the state of evaluation within the FPU and is set in circumstances where invalid operations, division by zero or overflow exceptions to the ANSI-IEEE 754-1985 standard would be flagged (page 14). *FP\_Error* is also set if an input to a floating point operation is infinite or is not a number (NaN). The *FP\_Error* flag can be set, tested and cleared without affecting the main *Error* flag, but can also set *Error* when required (page 14). Depending on how a program is compiled, it is possible for both unchecked and fully checked floating point arithmetic to be performed.

Further details on the operation of the FPU can be found in The Transputer Instruction Set - A Compiler Writers' Guide.

Table 5.2 Typical floating point operation times for IMS T800

Operation	T800-20		T800-30	
	Single length	Double length	Single length	Double length
add	350 ns	350 ns	233 ns	233 ns
subtract	350 ns	350 ns	233 ns	233 ns
multiply	550 ns	1000 ns	367 ns	667 ns
divide	850 ns	1600 ns	567 ns	1067 ns

Timing is for operations where both operands are normalised fp numbers.



## 6 System services

System services include all the necessary logic to initialise and sustain operation of the device. They also include error handling and analysis facilities.

### 6.1 Power

Power is supplied to the device via the **VCC** and **GND** pins. Several of each are provided to minimise inductance within the package. All supply pins must be connected. The supply must be decoupled close to the chip by at least one 100 nF low inductance (e.g. ceramic) capacitor between **VCC** and **GND**. Four layer boards are recommended; if two layer boards are used, extra care should be taken in decoupling.

Input voltages must not exceed specification with respect to **VCC** and **GND**, even during power-up and power-down ramping, otherwise *latchup* can occur. CMOS devices can be permanently damaged by excessive periods of latchup.

### 6.2 CapPlus, CapMinus

The internally derived power supply for internal clocks requires an external low leakage, low inductance 1  $\mu$ F capacitor to be connected between **CapPlus** and **CapMinus**. A ceramic capacitor is preferred, with an impedance less than 3 Ohms between 100 KHz and 10 MHz. If a polarised capacitor is used the negative terminal should be connected to **CapMinus**. Total PCB track length should be less than 50 mm. The connections must not touch power supplies or other noise sources.

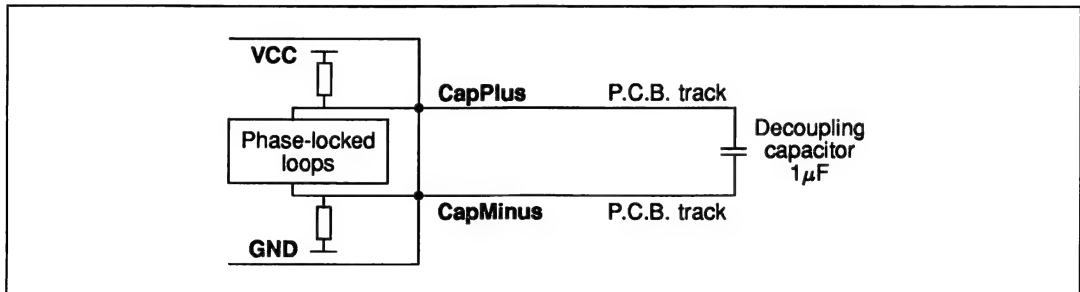


Figure 6.1 Recommended PLL decoupling

### 6.3 ClockIn

Transputer family components use a standard clock frequency, supplied by the user on the **ClockIn** input. The nominal frequency of this clock for all transputer family components is 5 MHz, regardless of device type, transputer word length or processor cycle time. High frequency internal clocks are derived from **ClockIn**, simplifying system design and avoiding problems of distributing high speed clocks externally.

A number of transputer devices may be connected to a common clock, or may have individual clocks providing each one meets the specified stability criteria. In a multi-clock system the relative phasing of **ClockIn** clocks is not important, due to the asynchronous nature of the links. Mark/space ratio is unimportant provided the specified limits of **ClockIn** pulse widths are met.

Oscillator stability is important. **ClockIn** must be derived from a crystal oscillator; RC oscillators are not sufficiently stable. **ClockIn** must not be distributed through a long chain of buffers. Clock edges must be monotonic and remain within the specified voltage and time limits.

Table 6.1 Input clock

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TDCLDCH	ClockIn pulse width low	40			ns	
TDCHDCL	ClockIn pulse width high	40			ns	
TDCLDCL	ClockIn period		200		ns	1,3
TDCerror	ClockIn timing error			±0.5	ns	2
TDC1DC2	Difference in ClockIn for 2 linked devices			400	ppm	3
TDCr	ClockIn rise time			10	ns	4
TDCf	ClockIn fall time			8	ns	4

**Notes**

- 1 Measured between corresponding points on consecutive falling edges.
- 2 Variation of individual falling edges from their nominal times.
- 3 This value allows the use of 200ppm crystal oscillators for two devices connected together by a link.
- 4 Clock transitions must be monotonic within the range **VIH** to **VIL** (table 11.3).

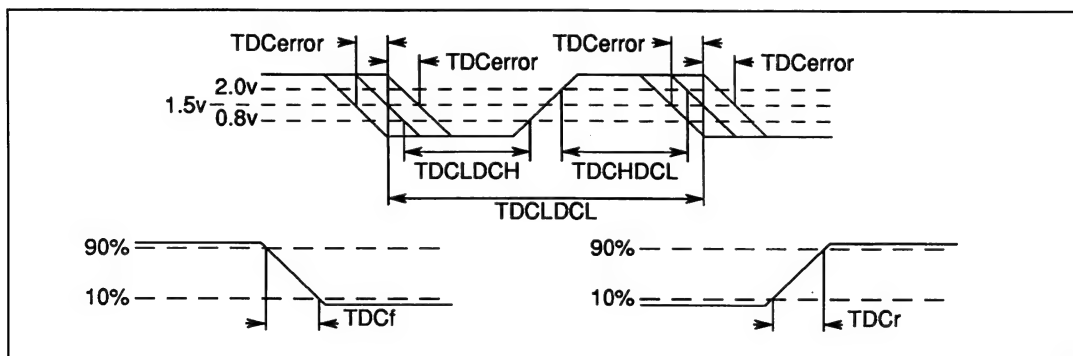


Figure 6.2 ClockIn timing

**6.4 ProcSpeedSelect0-2**

Processor speed of the IMS T800 is variable in discrete steps. The desired speed can be selected, up to the maximum rated for a particular component, by the three speed select lines **ProcSpeedSelect0-2**. The pins are tied high or low, according to the table below, for the various speeds. The **ProcSpeedSelect0-2** pins are designated **HoldToGND** on the IMS T414, and coding is so arranged that the IMS T800 can be plugged directly into a board designed for a 20 MHz IMS T414.

Only six of the possible speed select combinations are currently used; the other two are not valid speed selectors. The frequency of **ClockIn** for the speeds given in the table is 5 MHz.

Table 6.2 Processor speed selection

Proc Speed Select2	Proc Speed Select1	Proc Speed Select0	Processor Clock Speed MHz	Processor Cycle Time nS	Notes
0	0	0	20.0	50.0	
0	0	1	22.5	44.4	
0	1	0	25.0	40.0	
0	1	1	30.0	33.3	
1	0	0	35.0	28.6	
1	0	1			Invalid
1	1	0	17.5	57.1	
1	1	1			Invalid

Note: Inclusion of a speed selection in this table does not imply immediate availability.

## 6.5 Reset

**Reset** can go high with **VCC**, but must at no time exceed the maximum specified voltage for **VIH**. After **VCC** is valid **ClockIn** should be running for a minimum period **TDCVRL** before the end of **Reset**. The falling edge of **Reset** initialises the transputer, triggers the memory configuration sequence and starts the bootstrap routine. Link outputs are forced low during reset; link inputs and **EventReq** should be held low. Memory request (DMA) must not occur whilst **Reset** is high but can occur before bootstrap (page 50).

After the end of **Reset** there will be a delay of 144 periods of **ClockIn** (figure 6.3). Following this, the **MemWrD0**, **MemRfD1** and **MemAD2-31** pins will be scanned to check for the existence of a pre-programmed memory interface configuration (page 40). This lasts for a further 144 periods of **ClockIn**. Regardless of whether a configuration was found, 36 configuration read cycles will then be performed on external memory using the default memory configuration (page 41), in an attempt to access the external configuration ROM. A delay will then occur, its period depending on the actual configuration. Finally eight complete and consecutive refresh cycles will initialise any dynamic RAM, using the new memory configuration. If the memory configuration does not enable refresh of dynamic RAM the refresh cycles will be replaced by an equivalent delay with no external memory activity.

If **BootFromRom** is high bootstrapping will then take place immediately, using data from external memory; otherwise the transputer will await an input from any link. The processor will be in the low priority state.

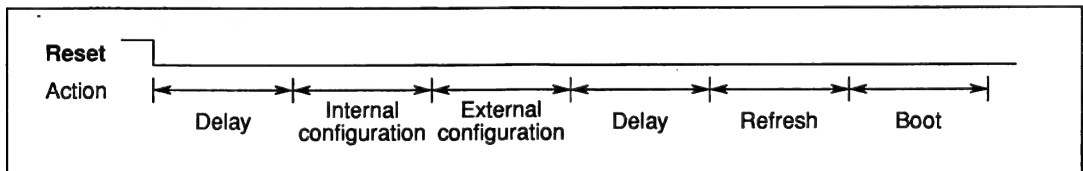


Figure 6.3 IMS T800 post-reset sequence

## 6.6 Bootstrap

The transputer can be bootstrapped either from a link or from external ROM. To facilitate debugging, **BootFromRom** may be dynamically changed but must obey the specified timing restrictions. It is sampled once only by the transputer, before the first instruction is executed after **Reset** is taken low.

If **BootFromRom** is connected high (e.g. to **VCC**) the transputer starts to execute code from the top two bytes in external memory, at address **#7FFFFFFE**. This location should contain a backward jump to a program in ROM. Following this access, **BootFromRom** may be taken low if required. The processor is in the low priority

state, and the *W* register points to *MemStart* (page 29).

Table 6.3 Reset and Analyse

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TPVRH	Power valid before Reset	10			ms	
TRHRL	Reset pulse width high	8			ClockIn ms	1
TDCVRL	ClockIn running before Reset end	10			ms	2
TAHRH	Analyse setup before Reset	3			ms	
TRLAL	Analyse hold after Reset end	1			ClockIn	1
TBRVRL	BootFromRom setup	0			ms	
TRLBRX	BootFromRom hold after Reset	0			ms	3
TALBRX	BootFromRom hold after Analyse					3

#### Notes

- 1 Full periods of **ClockIn** **TDCLDCL** required.
- 2 At power-on reset.
- 3 Must be stable until after end of bootstrap period. See Bootstrap section.

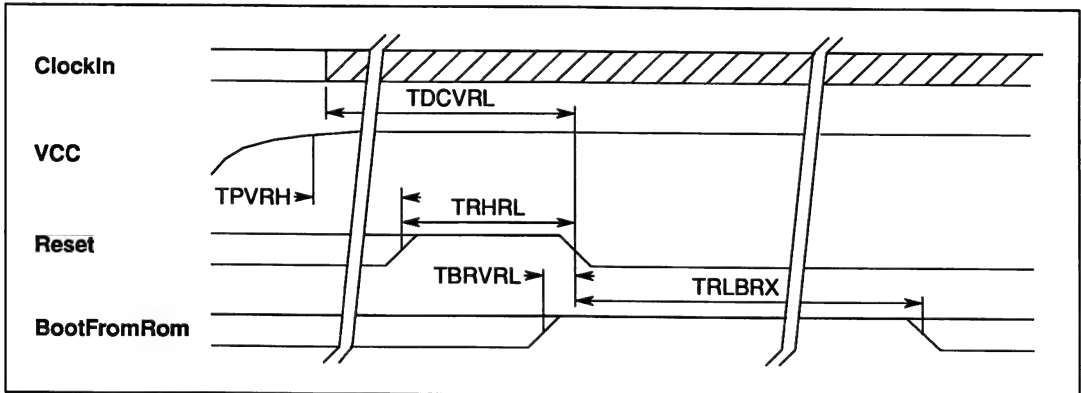


Figure 6.4 Transputer reset timing with Analyse low

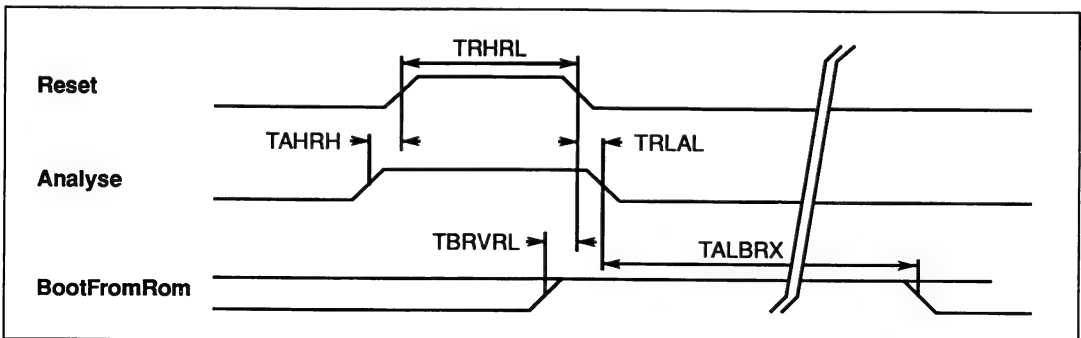


Figure 6.5 Transputer reset and analyse timing

If **BootFromRom** is connected low (e.g. to **GND**) the transputer will wait for the first bootstrap message to arrive on any one of its links. The transputer is ready to receive the first byte on a link within two processor cycles **TPCLPCL** after **Reset** goes low.

If the first byte received (the control byte) is greater than 1 it is taken as the quantity of bytes to be input. The following bytes, to that quantity, are then placed in internal memory starting at location *MemStart*. Following reception of the last byte the transputer will start executing code at *MemStart* as a low priority process. **BootFromRom** may be taken high after reception of the last byte, if required. The memory space immediately above the loaded code is used as work space. Messages arriving on other links after the control byte has been received and on the bootstrapping link after the last bootstrap byte will be retained until a process inputs from them.

## 6.7 Peek and poke

Any location in internal or external memory can be interrogated and altered when the transputer is waiting for a bootstrap from link. If the control byte is 0 then eight more bytes are expected on the same link. The first four byte word is taken as an internal or external memory address at which to poke (write) the second four byte word. If the control byte is 1 the next four bytes are used as the address from which to peek (read) a word of data; the word is sent down the output channel of the same link.

Following such a peek or poke, the transputer returns to its previously held state. Any number of accesses may be made in this way until the control byte is greater than 1, when the transputer will commence reading its bootstrap program. Any link can be used, but addresses and data must be transmitted via the same link as the control byte.

## 6.8 Analyse

If **Analyse** is taken high when the transputer is running, the transputer will halt at the next descheduling point (page 13). From **Analyse** being asserted, the processor will halt within three time slice periods plus the time taken for any high priority process to complete. As much of the transputer status is maintained as is necessary to permit analysis of the halted machine. Processor flags **Error** and **HaltOnError** are not altered at reset, whether **Analyse** is asserted or not. Memory refresh continues.

Input links will continue with outstanding transfers. Output links will not make another access to memory for data but will transmit only those bytes already in the link buffer. Providing there is no delay in link acknowledgement, the links should be inactive within a few microseconds of the transputer halting.

**Reset** should not be asserted before the transputer has halted and link transfers have ceased. When **Reset** is taken low whilst **Analyse** is high, neither the memory configuration sequence nor the block of eight refresh cycles will occur; the previous memory configuration will be used for any external memory accesses. If **BootFromRom** is high the transputer will bootstrap as soon as **Analyse** is taken low, otherwise it will await a control byte on any link. If **Analyse** is taken low without **Reset** going high the transputer state and operation are undefined. After the end of a valid **Analyse** sequence the registers have the values given in table 6.4.

Table 6.4 Register values after Analyse

<i>I</i>	<i>MemStart</i> if bootstrapping from a link, or the external memory bootstrap address if bootstrapping from ROM.
<i>W</i>	<i>MemStart</i> if bootstrapping from ROM, or the address of the first free word after the bootstrap program if bootstrapping from link.
<i>A</i>	The value of <i>I</i> when the processor halted.
<i>B</i>	The value of <i>W</i> when the processor halted, together with the priority of the process when the transputer was halted (i.e. the <i>W</i> descriptor).
<i>C</i>	The ID of the bootstrapping link if bootstrapping from link.

## 6.9 Error, ErrorIn

The **Error** pin carries the OR'ed output of the internal **Error** flag and the **ErrorIn** input. If **Error** is high it indicates either that **ErrorIn** is high or that an error was detected in one of the processes. An internal error can be caused, for example, by arithmetic overflow, divide by zero, array bounds violation or software setting the flag directly (page 14). It can also be set from the floating point unit under certain circumstances (page 14, 21). Once set, the **Error** flag is only cleared by executing the instruction *testerr*. The error is not cleared by processor reset, in order that analysis can identify any errant transputer (page 27).

A process can be programmed to stop if the **Error** flag is set; it cannot then transmit erroneous data to other processes, but processes which do not require that data can still be scheduled. Eventually all processes which rely, directly or indirectly, on data from the process in error will stop through lack of data. **ErrorIn** does not directly affect the status of a processor in any way.

By setting the **HaltOnError** flag the transputer itself can be programmed to halt if **Error** becomes set. If **Error** becomes set after **HaltOnError** has been set, all processes on that transputer will cease but will not necessarily cause other transputers in a network to halt. Setting **HaltOnError** after **Error** will not cause the transputer to halt; this allows the processor reset and analyse facilities to function with the flags in indeterminate states.

An alternative method of error handling is to have the errant process or transputer cause all transputers to halt. This can be done by 'daisy-chaining' the **ErrorIn** and **Error** pins of a number of processors and applying the final **Error** output signal to the **EventReq** pin of a suitably programmed master transputer. Since the process state is preserved when stopped by an error, the master transputer can then use the analyse function to debug the fault. When using such a circuit, note that the **Error** flag is in an indeterminate state on power up; the circuit and software should be designed with this in mind.

Error checks can be removed completely to optimise the performance of a proven program; any unexpected error then occurring will have an arbitrary undefined effect.

If a high priority process pre-empt a low priority one, status of the **Error** and **HaltOnError** flags is saved for the duration of the high priority process and restored at the conclusion of it. Status of both flags is transmitted to the high priority process. Either flag can be altered in the process without upsetting the error status of any complex operation being carried out by the pre-empted low priority process.

In the event of a transputer halting because of **HaltOnError**, the links will finish outstanding transfers before shutting down. If **Analyse** is asserted then all inputs continue but outputs will not make another access to memory for data. Memory refresh will continue to take place.

After halting due to the **Error** flag changing from 0 to 1 whilst **HaltOnError** is set, register *I* points two bytes past the instruction which set **Error**. After halting due to the **Analyse** pin being taken high, register *I* points one byte past the instruction being executed. In both cases *I* will be copied to register *A*.

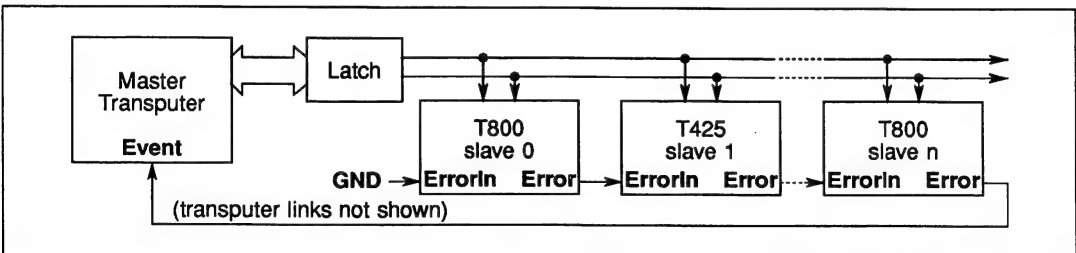


Figure 6.6 Error handling in a multi-transputer system

## 7 Memory

The IMS T800 has 4 Kbytes of fast internal static memory for high rates of data throughput. Each internal memory access takes one processor cycle **ProcClockOut** (page 31). The transputer can also access 4 Gbytes of external memory space. Internal and external memory are part of the same linear address space. Internal RAM can be disabled by holding **DisableIntRAM** high. All internal addresses are then mapped to external RAM. This pin should not be altered after **Reset** has been taken low.

IMS T800 memory is byte addressed, with words aligned on four-byte boundaries. The least significant byte of a word is the lowest addressed byte.

The bits in a byte are numbered 0 to 7, with bit 0 the least significant. The bytes are numbered from 0, with byte 0 the least significant. In general, wherever a value is treated as a number of component values, the components are numbered in order of increasing numerical significance, with the least significant component numbered 0. Where values are stored in memory, the least significant component value is stored at the lowest (most negative) address.

Internal memory starts at the most negative address **#80000000** and extends to **#80000FFF**. User memory begins at **#80000070**; this location is given the name *MemStart*.

A reserved area at the bottom of internal memory is used to implement link and event channels.

Two words of memory are reserved for timer use, *TPtrLoc0* for high priority processes and *TPtrLoc1* for low priority processes. They either indicate the relevant priority timer is not in use or point to the first process on the timer queue at that priority level.

Values of certain processor registers for the current low priority process are saved in the reserved *IntSaveLoc* locations when a high priority process pre-empt a low priority one. Other locations are reserved for extended features such as block moves and floating point operations.

External memory space starts at **#80001000** and extends up through **#00000000** to **#7FFFFFFF**. Memory configuration data and ROM bootstrapping code must be in the most positive address space, starting at **#7FFFFFF6C** and **#7FFFFFFE** respectively. Address space immediately below this is conventionally used for ROM based code.

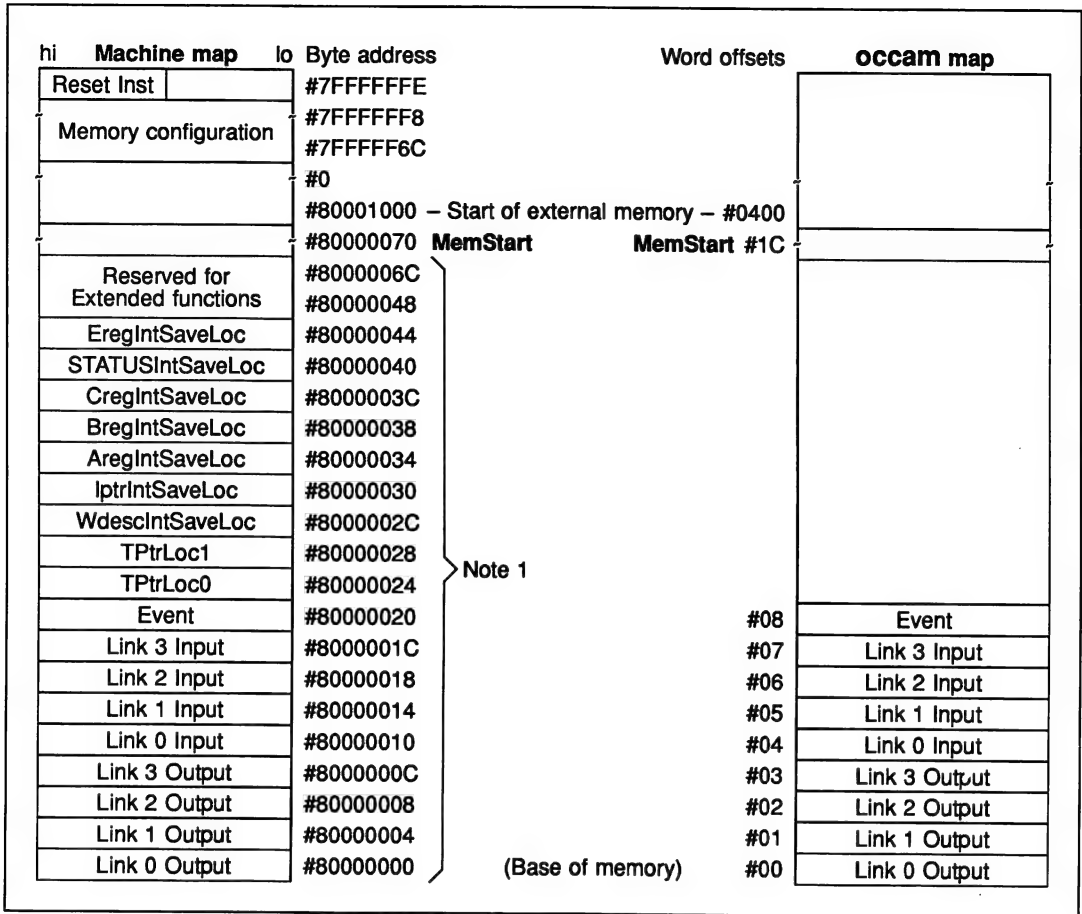


Figure 7.1 IMS T800 memory map

These locations are used as auxiliary processor registers and should not be manipulated by the user. Like processor registers, their contents may be useful for implementing debugging tools (**Analyse**, page 27). For details see *The Transputer Instruction Set - A Compiler Writers' Guide*.



## 8 External memory interface

The External Memory Interface (EMI) allows access to a 32 bit address space, supporting dynamic and static RAM as well as ROM and EPROM. EMI timing can be configured at **Reset** to cater for most memory types and speeds, and a program is supplied with the Transputer Development System to aid in this configuration.

There are 13 internal configurations which can be selected by a single pin connection (page 40). If none are suitable the user can configure the interface to specific requirements, as shown in page 41.

### 8.1 ProcClockOut

This clock is derived from the internal processor clock, which is in turn derived from **ClockIn**. Its period is equal to one internal microcode cycle time, and can be derived from the formula

$$TPCLPCL = TDCLDCL / PLLx$$

where **TPCLPCL** is the **ProcClockOut Period**, **TDCLDCL** is the **ClockIn Period** and **PLLx** is the phase lock loop factor for the relevant speed part, obtained from the ordering details (Ordering section).

The time value **Tm** is used to define the duration of **Tstates** and, hence, the length of external memory cycles; its value is exactly half the period of one **ProcClockOut** cycle ( $0.5 \cdot TPCLPCL$ ), regardless of mark/space ratio of **ProcClockOut**.

Edges of the various external memory strobes coincide with rising or falling edges of **ProcClockOut**. It should be noted, however, that there is a skew associated with each coincidence. The value of skew depends on whether coincidence occurs when the **ProcClockOut** edge and strobe edge are both rising, when both are falling or if either is rising when the other is falling. Timing values given in the strobe tables show the best and worst cases. If a more accurate timing relationship is required, the exact **Tstate** timing and strobe edge to **ProcClockOut** relationships should be calculated and the correct skew factors applied from the edge skew timing table 8.4.

The timing parameters in the following tables are based on full characterisation of the 17 MHz and 20 MHz parts. Data for higher speeds is based on tests on a limited number of samples and may change when full characterisation is completed.

### 8.2 Tstates

The external memory cycle is divided into six **Tstates** with the following functions:

- T1** Address setup time before address valid strobe.
- T2** Address hold time after address valid strobe.
- T3** Read cycle tristate or write cycle data setup.
- T4** Extendable data setup time.
- T5** Read or write data.
- T6** Data hold.

Under normal conditions each **Tstate** may be from one to four periods **Tm** long, the duration being set during memory configuration. The default condition on **Reset** is that all **Tstates** are the maximum four periods **Tm** long to allow external initialisation cycles to read slow ROM.

Period **T4** can be extended indefinitely by adding externally generated wait states.

An external memory cycle is always an even number of periods **Tm** in length and the start of **T1** always coincides with a rising edge of **ProcClockOut**. If the total configured quantity of periods **Tm** is an odd number, one extra period **Tm** will be added at the end of **T6** to force the start of the next **T1** to coincide with

a rising edge of **ProcClockOut**. This period is designated **E** in configuration diagrams (figure 8.11).

Table 8.1 ProcClockOut

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TPCLPCL	ProcClockOut period	<b>a-1</b>	<b>a</b>	<b>a+1</b>	ns	1
TPCHPCL	ProcClockOut pulse width high	<b>b-2.5</b>	<b>b</b>	<b>b+2.5</b>	ns	2
TPCLPCH	ProcClockOut pulse width low		<b>c</b>		ns	3
Tm	ProcClockOut half cycle	<b>b-0.5</b>	<b>b</b>	<b>b+0.5</b>	ns	2
TPCstab	ProcClockOut stability			4	%	4

#### Notes

- 1 **a** is **TDCLDCL/PLLx**.
- 2 **b** is  $0.5 \times \text{TPCLPCL}$  (half the processor clock period).
- 3 **c** is **TPCLPCL-TPCHPCL**.
- 4 Stability is the variation of cycle periods between two consecutive cycles, measured at corresponding points on the cycles.

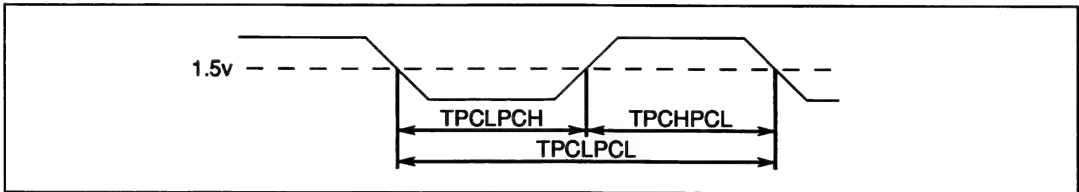


Figure 8.1 IMS T800 ProcClockOut timing

### 8.3 Internal access

During an internal memory access cycle the external memory interface bus **MemAD2-31** reflects the word address used to access internal RAM, **MemnotWrD0** reflects the read/write operation and **MemnotRfD1** is high; all control strobes are inactive. This is true unless and until a memory refresh cycle or DMA (memory request) activity takes place, when the bus will carry the appropriate external address or data.

The bus activity is not adequate to trace the internal operation of the transputer in full, but may be used for hardware debugging in conjunction with peek and poke (page 27).

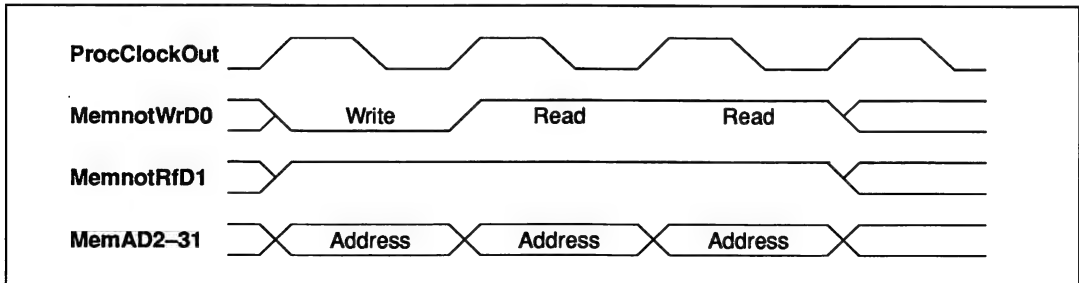


Figure 8.2 IMS T800 bus activity for internal memory cycle

#### 8.4 MemAD2-31

External memory addresses and data are multiplexed on one bus. Only the top 30 bits of address are output on the external memory interface, using pins **MemAD2-31**. They are normally output only during **Tstates T1** and **T2**, and should be latched during this time. Byte addressing is carried out internally by the transputer for read cycles. For write cycles the relevant bytes in memory are addressed by the write strobes **notMemWrB0-3**.

The data bus is 32 bits wide. It uses **MemAD2-31** for the top 30 bits and **MemnotRfD1** and **MemnotWrD0** for the lower two bits. Read cycle data may be set up on the bus at any time after the start of **T3**, but must be valid when the transputer reads it at the end of **T5**. Data may be removed any time during **T6**, but must be off the bus no later than the end of that period.

Write data is placed on the bus at the start of **T3** and removed at the end of **T6**. If **T6** is extended to force the next cycle **Tmx** (page 33) to start on a rising edge of **ProcClockOut**, data will be valid during this time also.

#### 8.5 MemnotWrD0

During **T1** and **T2** this pin will be low if the cycle is a write cycle, otherwise it will be high. During **Tstates T3** to **T6** it becomes bit 0 of the data bus. In both cases it follows the general timing of **MemAD2-31**.

#### 8.6 MemnotRfD1

During **T1** and **T2**, this pin is low if the address on **MemAD2-31** is a refresh address, otherwise it is high. During **Tstates T3** to **T6** it becomes bit 1 of the data bus. In both cases it follows the general timing of **MemAD2-31**.

#### 8.7 notMemRd

For a read cycle the read strobe **notMemRd** is low during **T4** and **T5**. Data is read by the transputer on the rising edge of this strobe, and may be removed immediately afterward. If the strobe duration is insufficient it may be extended by adding extra periods **Tm** to either or both of the **Tstates T4** and **T5**. Further extension may be obtained by inserting wait states at the end of **T4**.

In the read cycle timing diagrams **ProcClockOut** is included as a guide only; it is shown with each **Tstate** configured to one period **Tm**.

#### 8.8 notMemS0-4

To facilitate control of different types of memory and devices, the EMI is provided with five strobe outputs, four of which can be configured by the user. The strobes are conventionally assigned the functions shown in the read and write cycle diagrams, although there is no compulsion to retain these designations.

**notMemS0** is a fixed format strobe. Its leading edge is always coincident with the start of **T2** and its trailing edge always coincident with the end of **T5**.

The leading edge of **notMemS1** is always coincident with the start of **T2**, but its duration may be configured to be from zero to 31 periods **Tm**. Regardless of the configured duration, the strobe will terminate no later than the end of **T6**. The strobe is sometimes programmed to extend beyond the normal end of **Tmx**. When wait states are inserted into an EMI cycle the end of **Tmx** is delayed, but the potential active duration of the strobe is not altered. Thus the strobe can be configured to terminate relatively early under certain conditions (page 48). If **notMemS1** is configured to be zero it will never go low.

**notMemS2**, **notMemS3** and **notMemS4** are identical in operation. They all terminate at the end of **T5**, but the start of each can be delayed from one to 31 periods **Tm** beyond the start of **T2**. If the duration of one of these strobes would take it past the end of **T5** it will stay high. This can be used to cause a strobe to become active only when wait states are inserted. If one of these strobes is configured to zero it will never go low. Figure 8.5 shows the effect of **Wait** on strobes in more detail; each division on the scale is one period **Tm**.

Table 8.2 Read

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TaZdV	Address tristate to data valid	0			ns	
TdVRdH	Data setup before read	20			ns	
TRdHdX	Data hold after read	0			ns	
TS0LRdL	notMemS0 before start of read	a-2	a	a+2	ns	1
TS0HRdH	End of read from end of notMemS0	-1		1	ns	
TRdLRdH	Read period	b		b+6	ns	2

## Notes

- 1 **a** is total of **T2+T3** where **T2**, **T3** can be from one to four periods **Tm** each in length.
- 2 **b** is total of **T4+Twalt+T5** where **T4**, **T5** can be from one to four periods **Tm** each in length and **Twalt** may be any number of periods **Tm** in length.

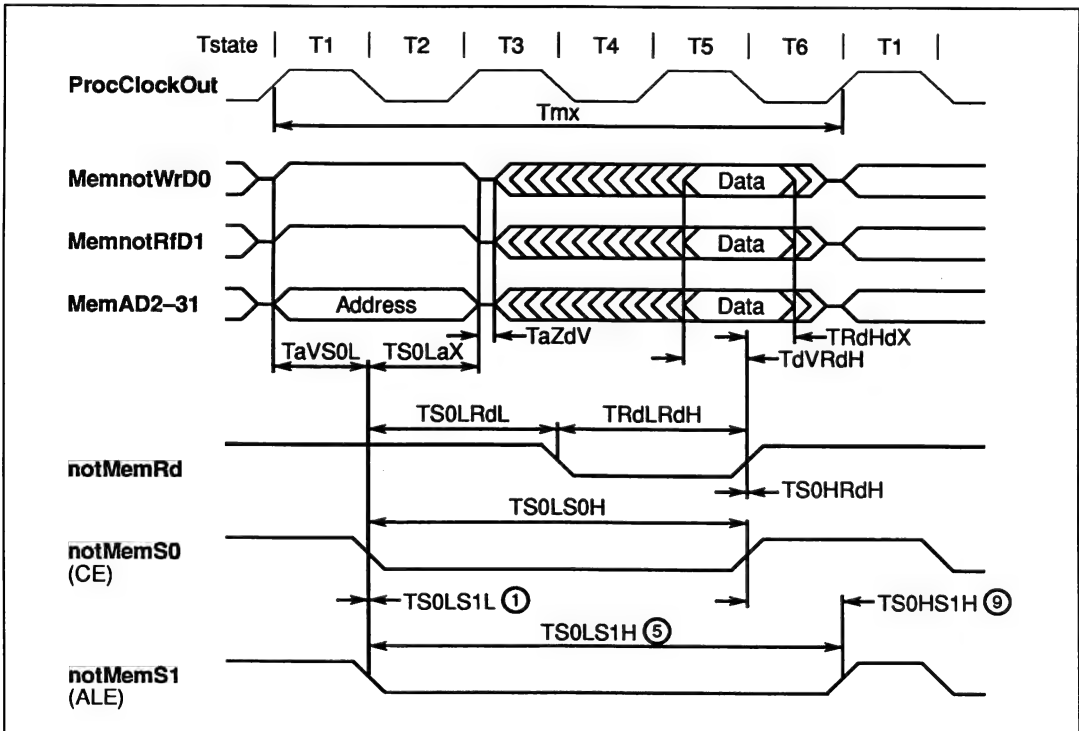


Figure 8.3 IMS T800 external read cycle: static memory

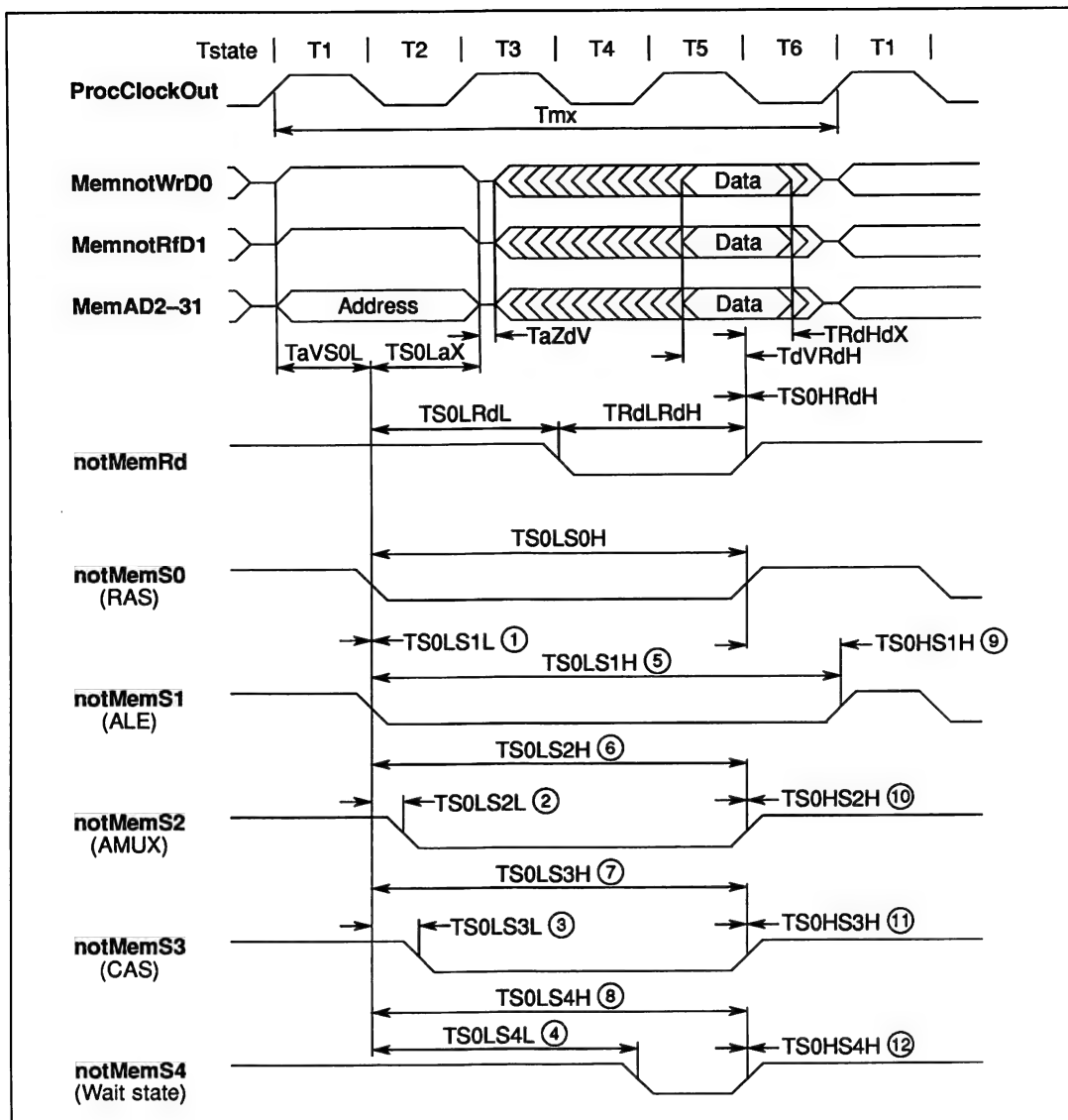


Figure 8.4 IMS T800 external read cycle: dynamic memory

Table 8.3 IMS T800 strobe timing

SYMBOL	(n)	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TaVS0L		Address setup before notMemS0		a		ns	1
TS0LaX		Address hold after notMemS0		b		ns	2
TS0LS0H		notMemS0 pulse width low	c		c+6	ns	3
TS0LS1L	1	notMemS1 from notMemS0	0		2	ns	
TS0LS1H	5	notMemS1 end from notMemS0	d		d+6	ns	4,6
TS0HS1H	9	notMemS1 end from notMemS0 end	e-1		e+4	ns	5,6
TS0LS2L	2	notMemS2 delayed after notMemS0	f-1		f+4	ns	7
TS0LS2H	6	notMemS2 end from notMemS0	c+4		c+8	ns	3
TS0HS2H	10	notMemS2 end from notMemS0 end	0		2	ns	
TS0LS3L	3	notMemS3 delayed after notMemS0	f-1		f+3	ns	7
TS0LS3H	7	notMemS3 end from notMemS0	c+4		c+8	ns	3
TS0HS3H	11	notMemS3 end from notMemS0 end	0		2	ns	
TS0LS4L	4	notMemS4 delayed after notMemS0	f-1		f+2	ns	7
TS0LS4H	8	notMemS4 end from notMemS0	c+4		c+8	ns	3
TS0HS4H	12	notMemS4 end from notMemS0 end	0		2	ns	
Tmx		Complete external memory cycle		g			8

## Notes

- 1 a is  $T_1$  where  $T_1$  can be from one to four periods  $T_m$  in length.
- 2 b is  $T_2$  where  $T_2$  can be from one to four periods  $T_m$  in length.
- 3 c is total of  $T_2+T_3+T_4+T_{wait}+T_5$  where  $T_2$ ,  $T_3$ ,  $T_4$ ,  $T_5$  can be from one to four periods  $T_m$  each in length and  $T_{wait}$  may be any number of periods  $T_m$  in length.
- 4 d can be from zero to 31 periods  $T_m$  in length.
- 5 e can be from -27 to +4 periods  $T_m$  in length.
- 6 If the configuration would cause the strobe to remain active past the end of  $T_6$  it will go high at the end of  $T_6$ . If the strobe is configured to zero periods  $T_m$  it will remain high throughout the complete cycle  $T_{mx}$ .
- 7 f can be from zero to 31 periods  $T_m$  in length. If this length would cause the strobe to remain active past the end of  $T_5$  it will go high at the end of  $T_5$ . If the strobe value is zero periods  $T_m$  it will remain low throughout the complete cycle  $T_{mx}$ .
- 8 g is one complete external memory cycle comprising the total of  $T_1+T_2+T_3+T_4+T_{wait}+T_5+T_6$  where  $T_1$ ,  $T_2$ ,  $T_3$ ,  $T_4$ ,  $T_5$  can be from one to four periods  $T_m$  each in length,  $T_6$  can be from one to five periods  $T_m$  in length and  $T_{wait}$  may be zero or any number of periods  $T_m$  in length.

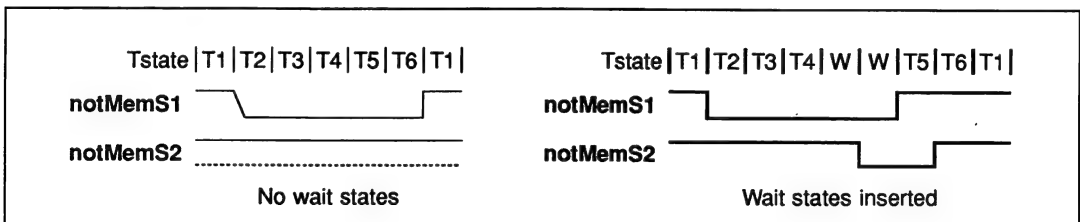


Figure 8.5 IMS T800 effect of wait states on strobes

Table 8.4 Strobe S0 to ProcClockOut skew

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TPCHS0H	Strobe rising from ProcClockOut rising	0		3	ns	
TPCLS0H	Strobe rising from ProcClockOut falling	1		4	ns	
TPCHS0L	Strobe falling from ProcClockOut rising	-3		0	ns	
TPCLS0L	Strobe falling from ProcClockOut falling	-1		2	ns	

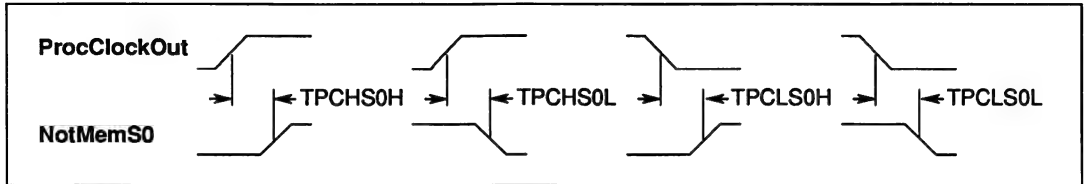


Figure 8.6 IMS T800 skew of notMemS0 to ProcClockOut

### 8.9 notMemWrB0-3

Because the transputer uses word addressing, four write strobes are provided; one to write each byte of the word. **notMemWrB0** addresses the least significant byte.

The transputer has both early and late write cycle modes. For a late write cycle the relevant write strobes **notMemWrB0-3** are low during **T4** and **T5**; for an early write they are also low during **T3**. Data should be latched into memory on the rising edge of the strobes in both cases, although it is valid until the end of **T6**. If the strobe duration is insufficient, it may be extended at configuration time by adding extra periods **Tm** to either or both of **Tstates** **T4** and **T5** for both early and late modes. For an early cycle they may also be added to **T3**. Further extension may be obtained by inserting wait states at the end of **T4**. If the data hold time is insufficient, extra periods **Tm** may be added to **T6** to extend it.

Table 8.5 Write

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TdVWrH	Data setup before write	<b>d</b>			ns	1,5
TWrHdX	Data hold after write	<b>a</b>			ns	1,2
TS0LWrL	notMemS0 before start of early write	<b>b-3</b>		<b>b+2</b>	ns	1,3
	notMemS0 before start of late write	<b>c-3</b>		<b>c+2</b>	ns	1,4
TS0HWrH	End of write from end of notMemS0	-2		2	ns	1
TWrLWrH	Early write pulse width	<b>d</b>		<b>d+6</b>	ns	1,5
	Late write pulse width	<b>e</b>		<b>e+6</b>	ns	1,6

#### Notes

- 1 Timing is for all write strobes **notMemWrB0-3**.
- 2 **a** is **T6** where **T6** can be from one to five periods **Tm** in length.
- 3 **b** is **T2** where **T2** can be from one to four periods **Tm** in length.
- 4 **c** is total of **T2+T3** where **T2**, **T3** can be from one to four periods **Tm** each in length.
- 5 **d** is total of **T3+T4+Twalt+T5** where **T3**, **T4**, **T5** can be from one to four periods **Tm** each in length and **Twalt** may be zero or any number of periods **Tm** in length.
- 6 **e** is total of **T4+Twalt+T5** where **T4**, **T5** can be from one to four periods **Tm** each in length and **Twalt** may be zero or any number of periods **Tm** in length.

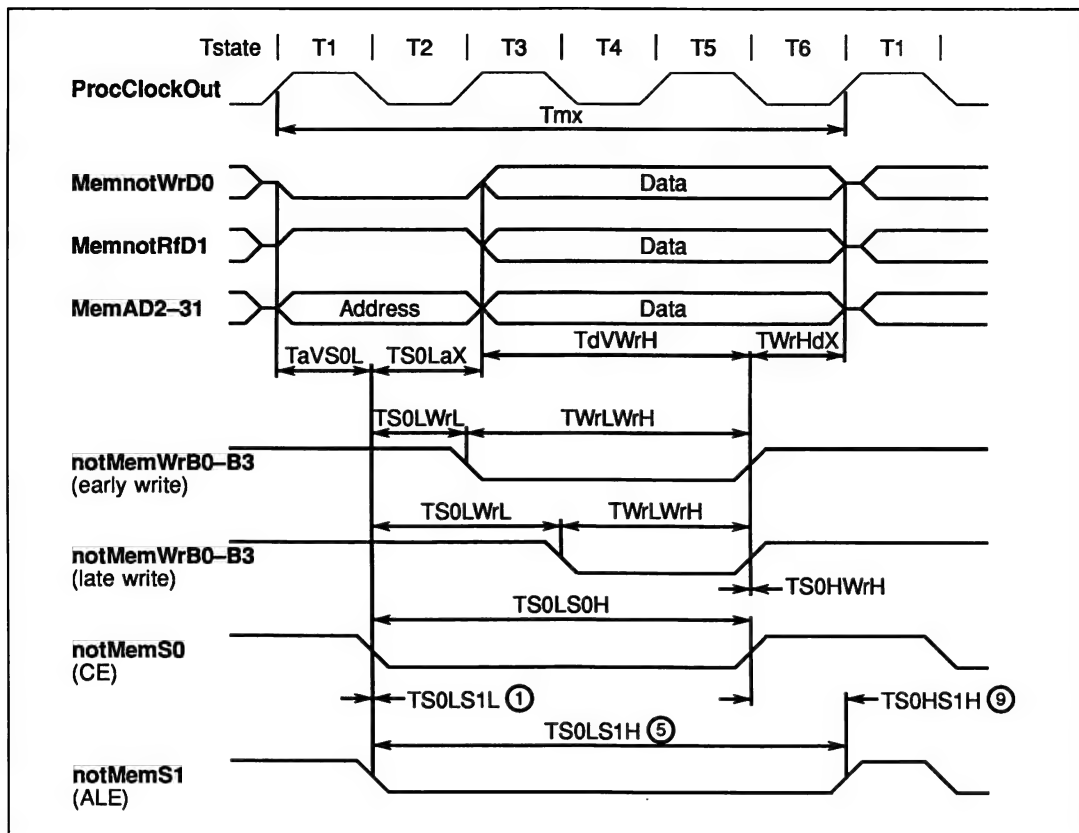


Figure 8.7 IMS T800 external write cycle

In the write cycle timing diagram **ProcClockOut** is included as a guide only; it is shown with each **Tstate** configured to one period **Tm**. The strobe is inactive during internal memory cycles.



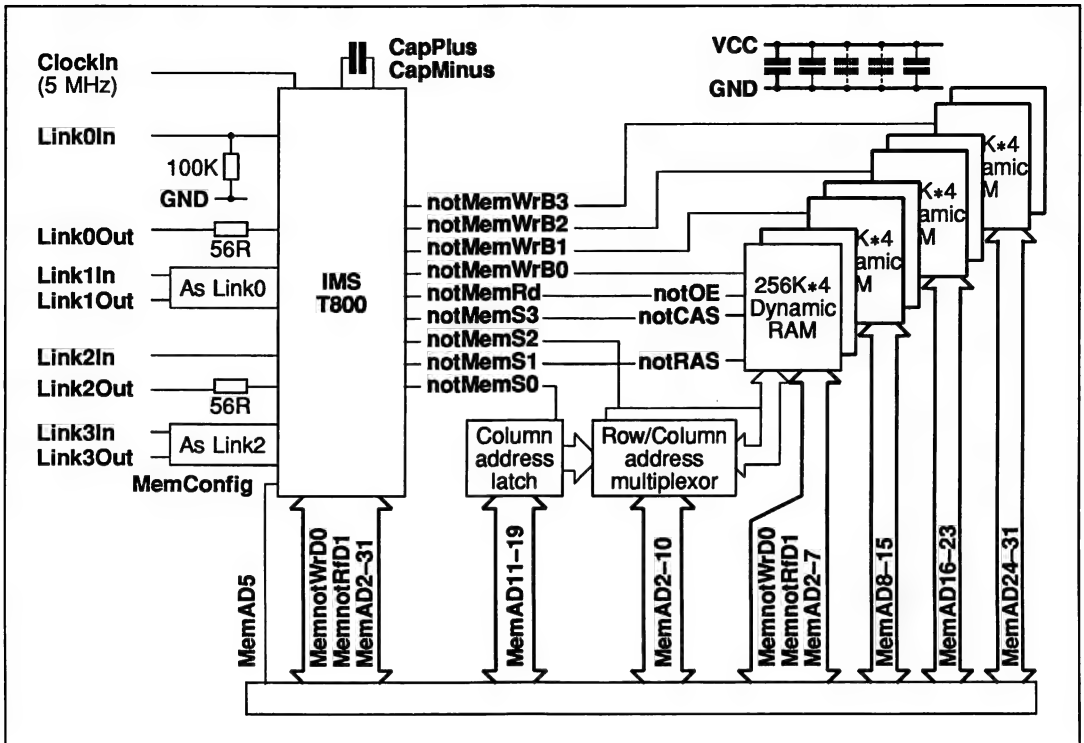


Figure 8.8 IMS T800 dynamic RAM application

## 8.10 MemConfig

**MemConfig** is an input pin used to read configuration data when setting external memory interface (EMI) characteristics. It is read by the processor on two occasions after **Reset** goes low; first to check if one of the preset internal configurations is required, then to determine a possible external configuration.

### 8.10.1 Internal configuration

The internal configuration scan comprises 64 periods **TDCLDCL** of **ClockIn** during the internal scan period of 144 **ClockIn** periods. **MemnotWrD0**, **MemnotRfD1** and **MemAD2-32** are all high at the beginning of the scan. Starting with **MemnotWrD0**, each of these lines goes low successively at intervals of two **ClockIn** periods and stays low until the end of the scan. If one of these lines is connected to **MemConfig** the preset internal configuration mode associated with that line will be used as the EMI configuration. The default configuration is that defined in the table for **MemAD31**; connecting **MemConfig** to **VCC** will also produce this default configuration. Note that only 17 of the possible configurations are valid, all others remain at the default configuration.

Table 8.6 IMS T800 internal configuration coding

Pin	Duration of each Tstate periods Tm						Strobe coefficient				Write cycle	Refresh interval	Cycle time
	T1	T2	T3	T4	T5	T6	s1	s2	s3	s4	type	ClockIn cycles	Proc cycles
<b>MemnotWrD0</b>	1	1	1	1	1	1	30	1	3	5	late	72	3
<b>MemnotRfD1</b>	1	2	1	1	1	2	30	1	2	7	late	72	4
<b>MemAD2</b>	1	2	1	1	2	3	30	1	2	7	late	72	5
<b>MemAD3</b>	2	3	1	1	2	3	30	1	3	8	late	72	6
<b>MemAD4</b>	1	1	1	1	1	1	3	1	2	3	early	72	3
<b>MemAD5</b>	1	1	2	1	2	1	5	1	2	3	early	72	4
<b>MemAD6</b>	2	1	2	1	3	1	6	1	2	3	early	72	5
<b>MemAD7</b>	2	2	2	1	3	2	7	1	3	4	early	72	6
<b>MemAD8</b>	1	1	1	1	1	1	30	1	2	3	early	†	3
<b>MemAD9</b>	1	1	2	1	2	1	30	2	5	9	early	†	4
<b>MemAD10</b>	2	2	2	2	4	2	30	2	3	8	late	72	7
<b>MemAD11</b>	3	3	3	3	3	3	30	2	4	13	late	72	9
<b>MemAD12</b>	1	1	2	1	2	1	4	1	2	3	early	72	4
<b>MemAD13</b>	2	1	2	1	2	2	5	1	2	3	early	72	5
<b>MemAD14</b>	2	2	2	1	3	2	6	1	3	4	early	72	6
<b>MemAD15</b>	2	1	2	3	3	3	8	1	2	3	early	72	7
<b>MemAD31</b>	4	4	4	4	4	4	31	30	30	18	late	72	12

† Provided for static RAM only.

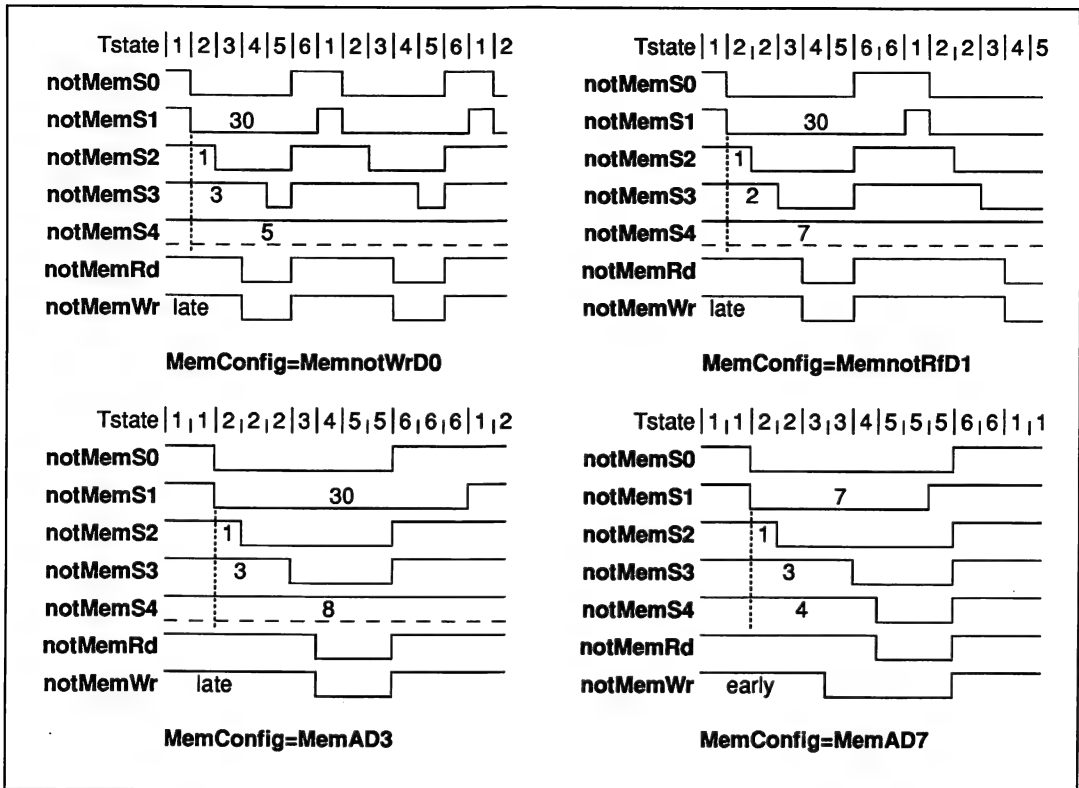


Figure 8.9 IMS T800 internal configuration

### 8.10.2 External configuration

If **MemConfig** is held low until **MemnotWrD0** goes low the internal configuration is ignored and an external configuration will be loaded instead. An external configuration scan always follows an internal one, but if an internal configuration occurs any external configuration is ignored.

The external configuration scan comprises 36 successive external read cycles, using the default EMI configuration preset by **MemAD31**. However, instead of data being read on the data bus as for a normal read cycle, only a single bit of data is read on **MemConfig** at each cycle. Addresses put out on the bus for each read cycle are shown in table 8.7, and are designed to address ROM at the top of the memory map. The table shows the data to be held in ROM; data required at the **MemConfig** pin is the inverse of this.

**MemConfig** is typically connected via an inverter to **MemnotWrD0**. Data bit zero of the least significant byte of each ROM word then provides the configuration data stream. By switching **MemConfig** between various data bus lines up to 32 configurations can be stored in ROM, one per bit of the data bus. **MemConfig** can be permanently connected to a data line or to GND. Connecting **MemConfig** to GND gives all **Tstates** configured to four periods; **notMemS1** pulse of maximum duration; **notMemS2-4** delayed by maximum; refresh interval 72 periods of **ClockIn**; refresh enabled; late write.

The external memory configuration table 8.7 shows the contribution of each memory address to the 13 configuration fields. The lowest 12 words (#7FFFFFFC to #7FFFFFF98, fields 1 to 6) define the number of extra periods **Tm** to be added to each **Tstate**. If field 2 is 3 then three extra periods will be added to **T2** to extend it to the maximum of four periods.

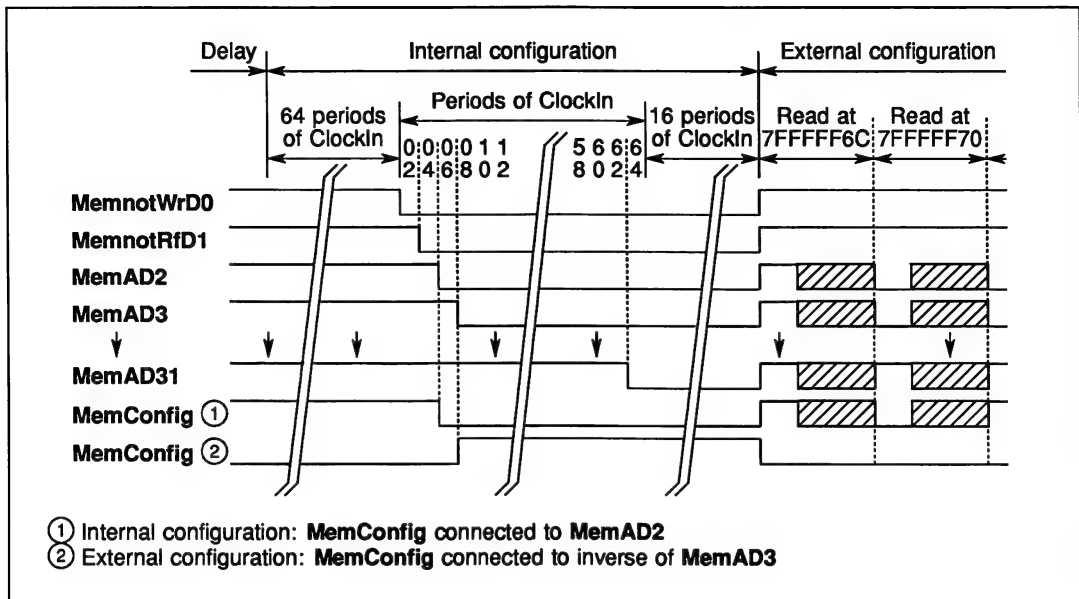


Figure 8.10 IMS T800 internal configuration scan

The next five addresses (field 7) define the duration of **notMemS1** and the following fifteen (fields 8 to 10) define the delays before strobes **notMemS2-4** become active. The five bits allocated to each strobe allow durations of from 0 to 31 periods **Tm**, as described in strobes page 33.

Addresses #7FFFFFFEC to #7FFFFFFF4 (fields 11 and 12) define the refresh interval and whether refresh is to be used, whilst the final address (field 13) supplies a high bit to **MemConfig** if a late write cycle is required.

The columns to the right of the coding table show the values of each configuration bit for the four sample external configuration diagrams. Note the inclusion of period **E** at the end of **T6** in some diagrams. This is inserted to bring the start of the next **Tstate T1** to coincide with a rising edge of **ProcClockOut** (page 31).

Wait states **W** have been added to show the effect of them on strobe timing; they are not part of a configuration. In each case which includes wait states, two wait periods are defined. This shows that if a wait state would cause the start of **T5** to coincide with a falling edge of **ProcClockOut**, another period **Tm** is generated by the EMI to force it to coincide with a rising edge of **ProcClockOut**. This coincidence is only necessary if wait states are added, otherwise coincidence with a falling edge is permitted. Any configuration memory access is only permitted to be extended using wait, up to a total of 14 **ClockIn** periods.

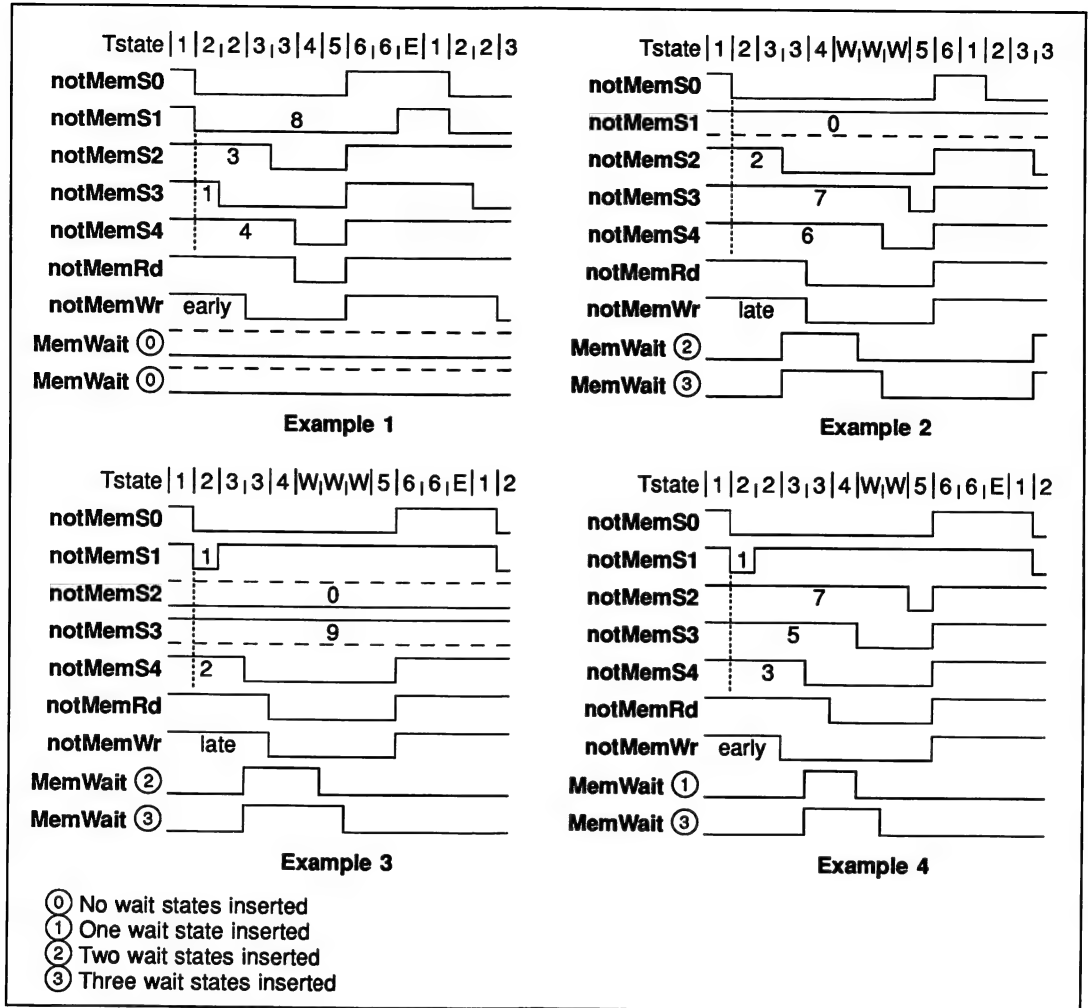


Figure 8.11 IMS T800 external configuration

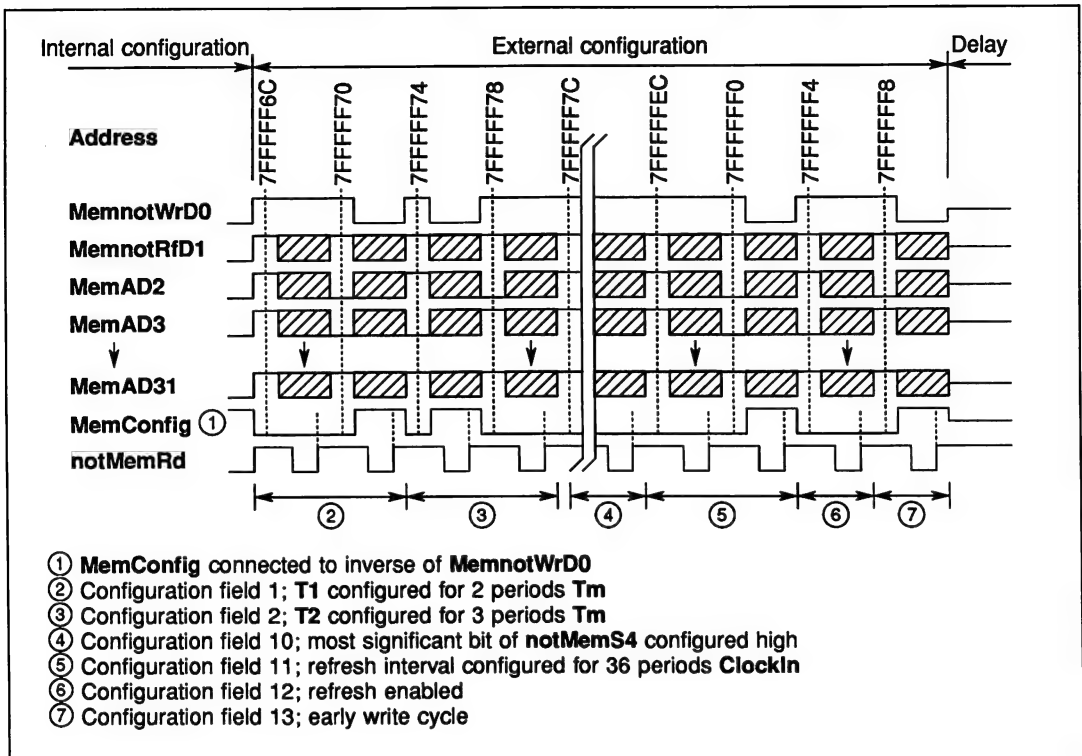


Figure 8.12 IMS T800 external configuration scan

Table 8.7 IMS T800 external configuration coding

Scan cycle	MemAD address	Field	Function	Example diagram			
				1	2	3	4
1	7FFFFFF6C	1	T1 least significant bit	0	0	0	0
2	7FFFFFF70	1	T1 most significant bit	0	0	0	0
3	7FFFFFF74	2	T2 least significant bit	1	0	0	1
4	7FFFFFF78	2	T2 most significant bit	0	0	0	0
5	7FFFFFF7C	3	T3 least significant bit	1	1	1	1
6	7FFFFFF80	3	T3 most significant bit	0	0	0	0
7	7FFFFFF84	4	T4 least significant bit	0	0	0	0
8	7FFFFFF88	4	T4 most significant bit	0	0	0	0
9	7FFFFFF8C	5	T5 least significant bit	0	0	0	0
10	7FFFFFF90	5	T5 most significant bit	0	0	0	0
11	7FFFFFF94	6	T6 least significant bit	1	0	1	1
12	7FFFFFF98	6	T6 most significant bit	0	0	0	0
13	7FFFFFF9C	7	notMemS1 least significant bit	0	0	1	1
14	7FFFFFFA0	7		0	0	0	0
15	7FFFFFFA4	7	↓ ↓	0	0	0	0
16	7FFFFFFA8	7		1	0	0	0
17	7FFFFFFAC	7	notMemS1 most significant bit	0	0	0	0
18	7FFFFFFB0	8	notMemS2 least significant bit	1	0	0	1
19	7FFFFFFB4	8		1	1	0	1
20	7FFFFFFB8	8	↓ ↓	0	0	0	1
21	7FFFFFFBC	8		0	0	0	0
22	7FFFFFFC0	8	notMemS2 most significant bit	0	0	0	0
23	7FFFFFFC4	9	notMemS3 least significant bit	1	1	1	1
24	7FFFFFFC8	9		0	1	0	0
25	7FFFFFFCC	9	↓ ↓	0	1	0	1
26	7FFFFFFD0	9		0	0	1	0
27	7FFFFFFD4	9	notMemS3 most significant bit	0	0	0	0
28	7FFFFFFD8	10	notMemS4 least significant bit	0	0	0	1
29	7FFFFFFDC	10		0	1	1	1
30	7FFFFFFE0	10	↓ ↓	1	1	0	0
31	7FFFFFFE4	10		0	0	0	0
32	7FFFFFFE8	10	notMemS4 most significant bit	0	0	0	0
33	7FFFFFFEC	11	Refresh Interval least significant bit	-	-	-	-
34	7FFFFFFF0	11	Refresh Interval most significant bit	-	-	-	-
35	7FFFFFFF4	12	Refresh Enable	-	-	-	-
36	7FFFFFFF8	13	Late Write	0	1	1	0





### 8.11 notMemRf

The IMS T800 can be operated with memory refresh enabled or disabled. The selection is made during memory configuration, when the refresh interval is also determined. Refresh cycles do not interrupt internal memory accesses, although the internal addresses cannot be reflected on the external bus during refresh.

When refresh is disabled no refresh cycles occur. During the post-Reset period eight dummy refresh cycles will occur with the appropriate timing but with no bus or strobe activity.

A refresh cycle uses the same basic external memory timing as a normal external memory cycle, except that it starts two periods  $T_m$  before the start of  $T_1$ . If a refresh cycle is due during an external memory access, it will be delayed until the end of that external cycle. Two extra periods  $T_m$  (periods  $R$  in the diagram) will then be inserted between the end of  $T_6$  of the external memory cycle and the start of  $T_1$  of the refresh cycle itself. The refresh address and various external strobes become active approximately one period  $T_m$  before  $T_1$ . Bus signals are active until the end of  $T_2$ , whilst **notMemRf** remains active until the end of  $T_6$ .

For a refresh cycle, **MemnotRfD1** goes low before **notMemRf** goes low and **MemnotWrD0** goes high with the same timing as **MemnotRfD1**. All the address lines share the same timing, but only **MemAD2-11** give the refresh address. **MemAD12-30** stay high during the address period, whilst **MemAD31** remains low. Refresh cycles generate strobes **notMemS0-4** with timing as for a normal external cycle, but **notMemRd** and **notMemWrB0-3** remain high. **MemWait** operates normally during refresh cycles.

Table 8.10 Memory refresh

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TRfLRfH	Refresh pulse width low	<b>a</b>		<b>a+6</b>	ns	1
TRaVSOL	Refresh address setup before notMemS0		<b>b</b>		ns	2
TRfLSOL	Refresh indicator setup before notMemS0		<b>b</b>		ns	2

**Notes**

- 1 **a** is total  $T_{mx} + T_m$ .
- 2 **b** is total  $T_1 + T_m$  where  $T_1$  can be from one to four periods  $T_m$  in length.

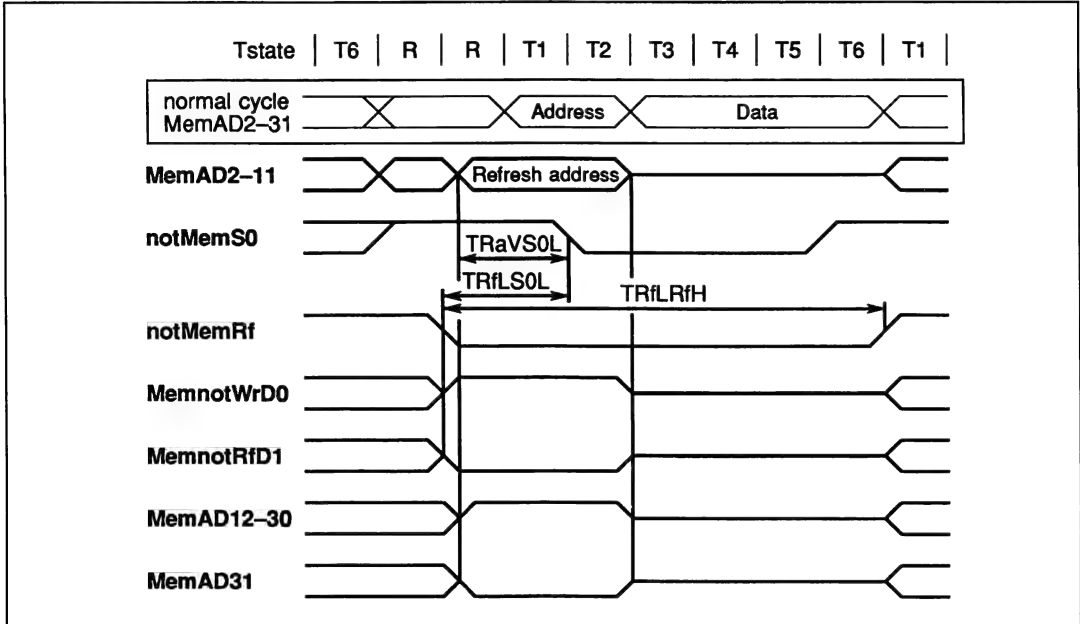


Figure 8.14 IMS T800 refresh cycle timing

**8.12 MemWait**

Taking **MemWait** high with the timing shown will extend the duration of **T4**. **MemWait** is sampled relative to the falling edge of **ProcClockOut** during a **T3** period, and should not change state in this region. By convention, **notMemS4** is used to synchronize wait state insertion. If this or another strobe is used, its delay should be such as to take the strobe low an even number of periods  $T_m$  after the start of **T1**, to coincide with a rising edge of **ProcClockOut**.

**MemWait** may be kept high indefinitely, although if dynamic memory refresh is used it should not be kept high long enough to interfere with refresh timing. **MemWait** operates normally during all cycles, including refresh and configuration cycles. It does not affect internal memory access in any way.

If the start of **T5** would coincide with a falling edge of **ProcClockOut** an extra wait period  $T_m$  (**EW**) is generated by the EMI to force coincidence with a rising edge. Rising edge coincidence is only forced if wait states are added, otherwise coincidence with a falling edge is permitted.

Table 8.11 Memory wait

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TPCHWtH	Wait setup	$0.5T_m+3$			ns	1,2
TPCHWtL	Wait hold	$0.5T_m+3$			ns	1,2
TWtLWtH	Delay before re-assertion of Wait	$2T_m$			ns	

Notes

- 1 ProcClockOut load should not exceed 50pf.
- 2 If wait period exceeds refresh interval, refresh cycles will be lost.

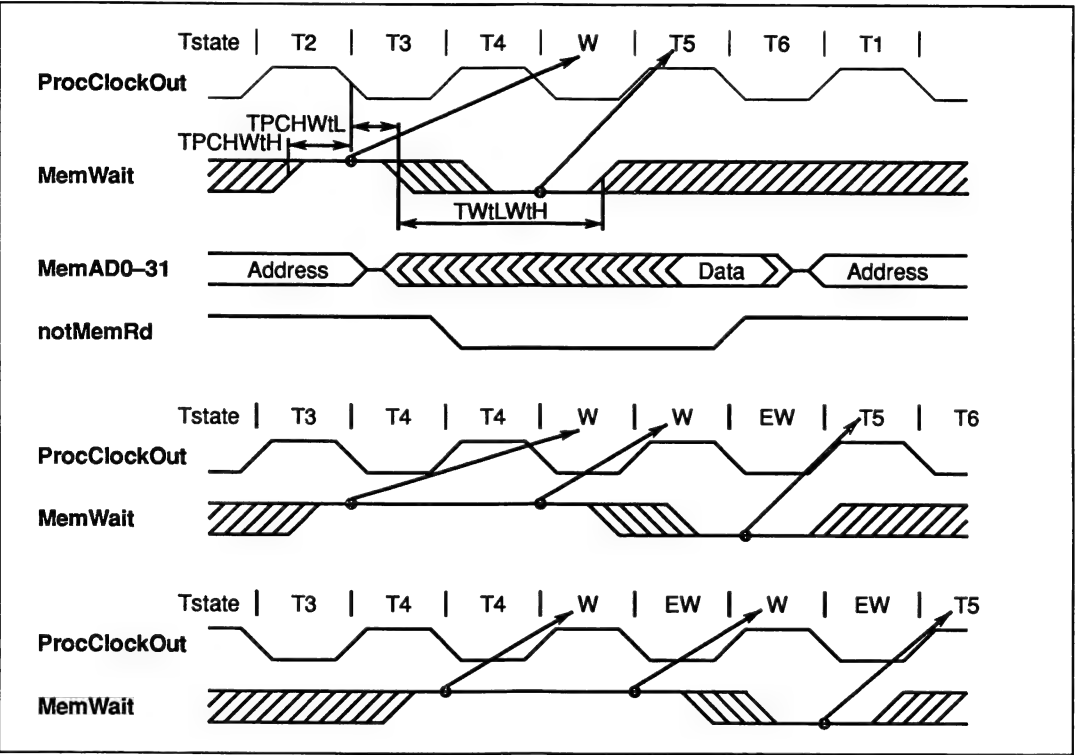


Figure 8.15 IMS T800 memory wait timing

### 8.13 MemReq, MemGranted

Direct memory access (DMA) can be requested at any time by taking the asynchronous **MemReq** input high. The transputer samples **MemReq** during the final period **T<sub>m</sub>** of **T<sub>6</sub>** of both refresh and external memory cycles. To guarantee taking over the bus immediately following either, **MemReq** must be set up at least two periods **T<sub>m</sub>** before the end of **T<sub>6</sub>**. In the absence of an external memory cycle, **MemReq** is sampled during every low period of **ProcClockOut**. The address bus is tristated two periods **T<sub>m</sub>** after the **ProcClockOut** rising edge which follows the sample. **MemGranted** is asserted one period **T<sub>m</sub>** after that.

Removal of **MemReq** is sampled during each low period of **ProcClockOut** and **MemGranted** is removed synchronously with the next falling edge of **ProcClockOut**. If accurate timing of DMA is required, **MemReq** should be set low coincident with a falling edge of **ProcClockOut**. Further external bus activity, either refresh, external cycles or reflection of internal cycles, will commence at the next rising edge of **ProcClockOut**.

Strobes are left in their inactive states during DMA. DMA cannot interrupt a refresh or external memory cycle, and outstanding refresh cycles will occur before the bus is released to DMA. DMA does not interfere with internal memory cycles in any way, although a program running in internal memory would have to wait for the end of DMA before accessing external memory. DMA cannot access internal memory. If DMA extends longer than one refresh interval (Memory Refresh Configuration Coding, table 8.8), the DMA user becomes responsible for refresh. DMA may also inhibit an internally running program from accessing external memory.

DMA allows a bootstrap program to be loaded into external RAM ready for execution after reset. If **MemReq** is held high throughout reset, **MemGranted** will be asserted before the bootstrap sequence begins. **MemReq** must be high at least one period **TDCLDCL** of **ClockIn** before **Reset**. The circuit should be designed to ensure correct operation if **Reset** could interrupt a normal DMA cycle.

Table 8.12 Memory request

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TMRHMGH	Memory request response time	4		7	T <sub>m</sub>	1
TMRLMGL	Memory request end response time	2		5	T <sub>m</sub>	
TADZMGH	Bus tristate before memory granted		1		T <sub>m</sub>	
TMGLADV	Bus active after end of memory granted		1		T <sub>m</sub>	

#### Notes

- 1 These values assume no external memory cycle is in progress. If an external cycle is active, maximum time could be (1 EMI cycle **T<sub>mx</sub>**)+(1 refresh cycle **TRfLRfH**)+(6 periods **T<sub>m</sub>**).

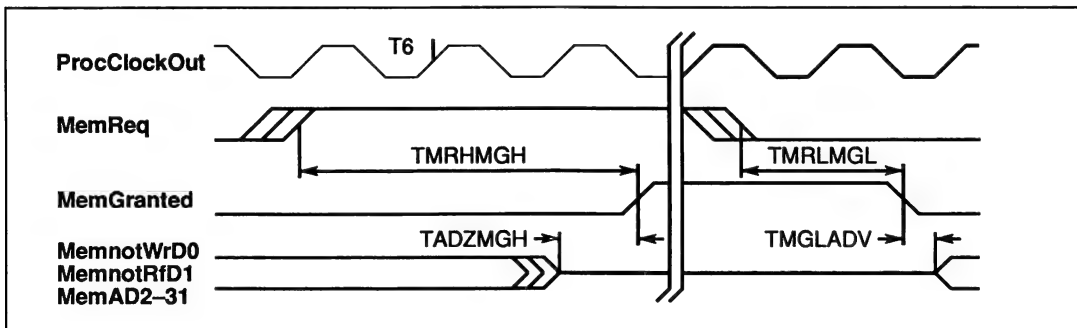


Figure 8.16 IMS T800 memory request timing

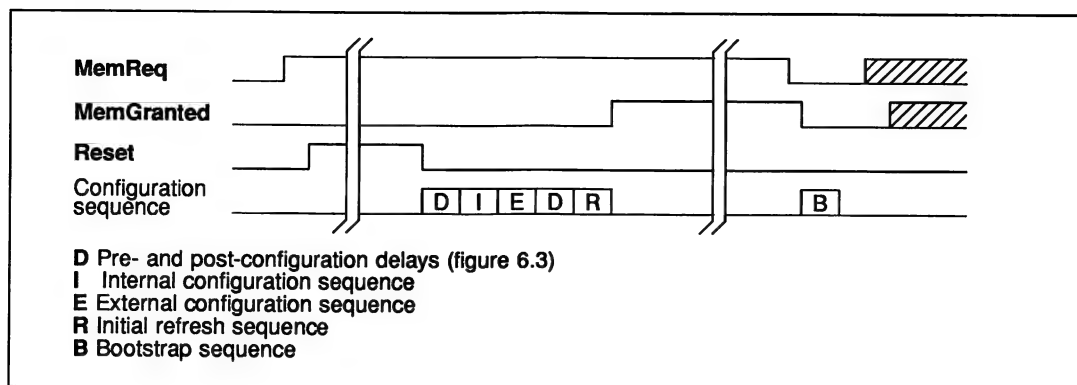


Figure 8.17 IMS T800 DMA sequence at reset

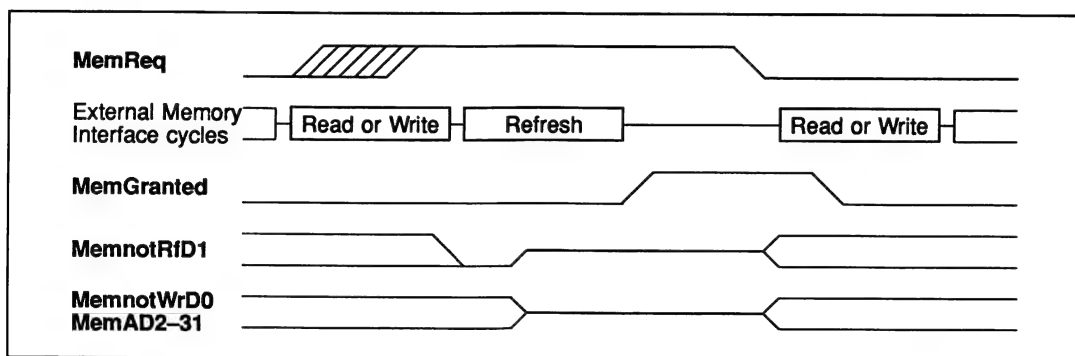


Figure 8.18 IMS T800 operation of MemReq, MemGranted with external, refresh memory cycles

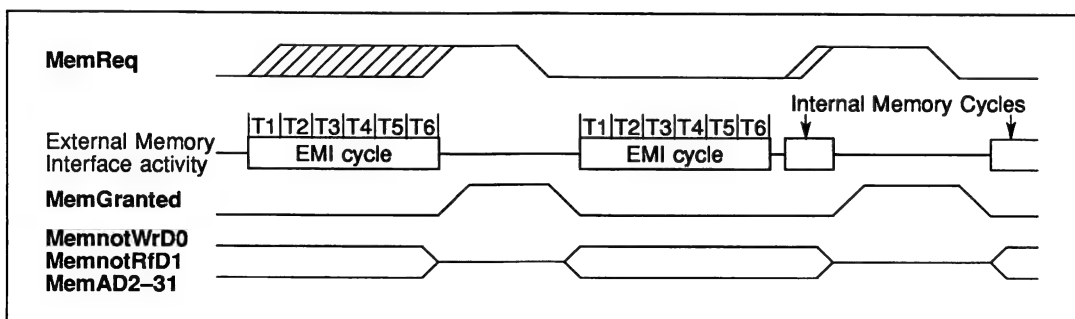


Figure 8.19 IMS T800 operation of MemReq, MemGranted with external, internal memory cycles

## 9 Events

**EventReq** and **EventAck** provide an asynchronous handshake interface between an external event and an internal process. When an external event takes **EventReq** high the external event channel (additional to the external link channels) is made ready to communicate with a process. When both the event channel and the process are ready the processor takes **EventAck** high and the process, if waiting, is scheduled. **EventAck** is removed after **EventReq** goes low.

Only one process may use the event channel at any given time. If no process requires an event to occur **EventAck** will never be taken high. Although **EventReq** triggers the channel on a transition from low to high, it must not be removed before **EventAck** is high. **EventReq** should be low during **Reset**; if not it will be ignored until it has gone low and returned high. **EventAck** is taken low when **Reset** occurs.

If the process is a high priority one and no other high priority process is running, the latency is as described on page 10. Setting a high priority task to wait for an event input is a way of interrupting a transputer program.

Table 9.1 Event

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TVHKH	Event request response	0			ns	
TKHVL	Event request hold	0			ns	
TVLKL	Delay before removal of event acknowledge	0		a	ns	1
TKLVH	Delay before re-assertion of event request	0			ns	
TKHEWL	Event acknowledge to end of event waiting	0			ns	
TKLEWH	End of event acknowledge to event waiting	0			ns	

### Notes

1 a is 3 processor cycles **TPCLPCL**.

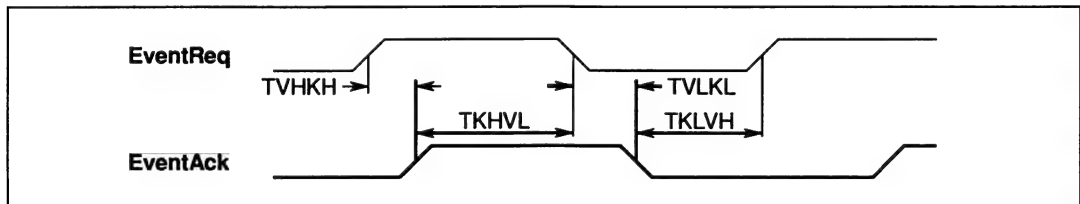


Figure 9.1 IMS T800 event timing

## 10 Links

Four identical INMOS bi-directional serial links provide synchronized communication between processors and with the outside world. Each link comprises an input channel and output channel. A link between two transputers is implemented by connecting a link interface on one transputer to a link interface on the other transputer. Every byte of data sent on a link is acknowledged on the input of the same link, thus each signal line carries both data and control information.

The quiescent state of a link output is low. Each data byte is transmitted as a high start bit followed by a one bit followed by eight data bits followed by a low stop bit. The least significant bit of data is transmitted first. After transmitting a data byte the sender waits for the acknowledge, which consists of a high start bit followed by a zero bit. The acknowledge signifies both that a process was able to receive the acknowledged data byte and that the receiving link is able to receive another byte. The sending link reschedules the sending process only after the acknowledge for the final byte of the message has been received.

The IMS T800 links allow an acknowledge packet to be sent before the data packet has been fully received. This overlapped acknowledge technique is fully compatible with all other INMOS transputer links.

The IMS T800 links support the standard INMOS communication speed of 10 Mbits/sec. In addition they can be used at 5 or 20 Mbits/sec. Links are not synchronised with **ClockIn** or **ProcClockOut** and are insensitive to their phases. Thus links from independently clocked systems may communicate, providing only that the clocks are nominally identical and within specification.

Links are TTL compatible and intended to be used in electrically quiet environments, between devices on a single printed circuit board or between two boards via a backplane. Direct connection may be made between devices separated by a distance of less than 300 millimetres. For longer distances a matched 100 Ohm transmission line should be used with series matching resistors **RM**. When this is done the line delay should be less than 0.4 bit time to ensure that the reflection returns before the next data bit is sent.

Buffers may be used for very long transmissions. If so, their overall propagation delay should be stable within the skew tolerance of the link, although the absolute value of the delay is immaterial.

Link speeds can be set by **LinkSpecial**, **Link0Special** and **Link123Special**. The link 0 speed can be set independently. Table 10.1 shows uni-directional and bi-directional data rates in Kbytes/sec for each link speed; **LinknSpecial** is to be read as **Link0Special** when selecting link 0 speed and as **Link123Special** for the others. Data rates are quoted for a transputer using internal memory, and will be affected by a factor depending on the number of external memory accesses and the length of the external memory cycle.

Table 10.1 Speed Settings for Transputer Links

Link Special	Linkn Special	Mbits/sec	Kbytes/sec	
			Uni	Bi
0	0	10	910	1250
0	1	5	450	670
1	0	10	910	1250
1	1	20	1740	2350

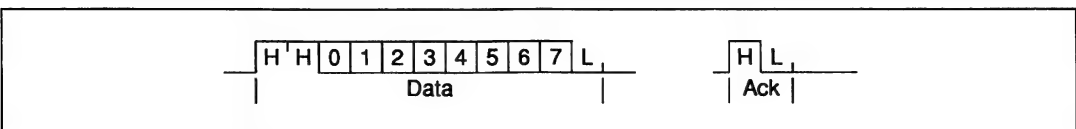


Figure 10.1 IMS T800 link data and acknowledge packets

Table 10.2 Link

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
TJQr	LinkOut rise time			20	ns	
TJQf	LinkOut fall time			10	ns	
TJDr	LinkIn rise time			20	ns	
TJDf	LinkIn fall time			20	ns	
TJQJD	Buffered edge delay	0			ns	
TJBskew	Variation in TJQJD			3	ns	1
				10	ns	1
				30	ns	1
CLIZ	LinkIn capacitance			7	pF	
CLL	LinkOut load capacitance			50	pF	
RM	Series resistor for 100Ω transmission line		56		ohms	

Notes

- 1 This is the variation in the total delay through buffers, transmission lines, differential receivers etc., caused by such things as short term variation in supply voltages and differences in delays for rising and falling edges.

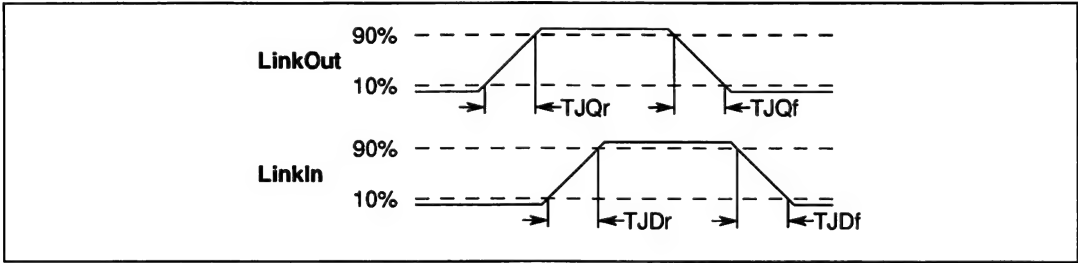


Figure 10.2 IMS T800 link timing

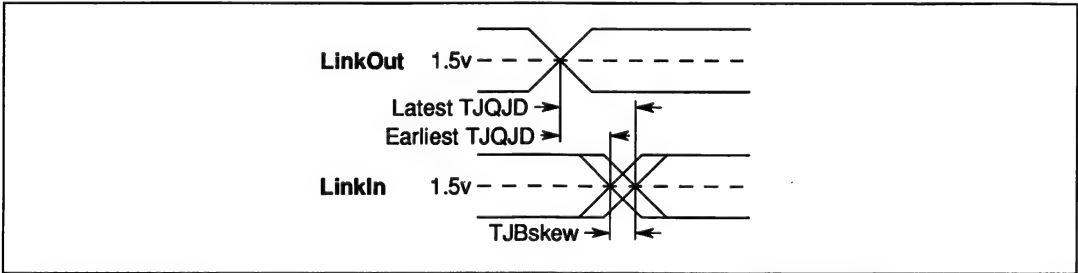


Figure 10.3 IMS T800 buffered link timing



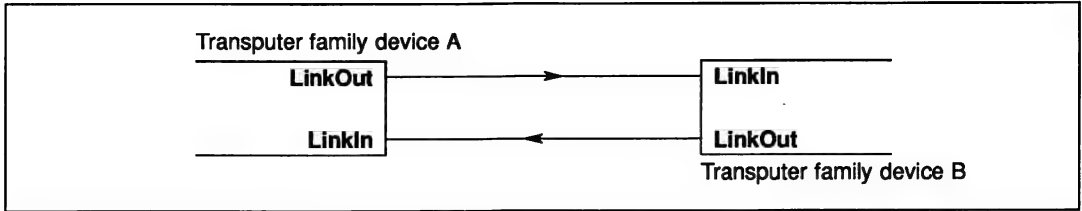


Figure 10.4 IMS T800 Links directly connected

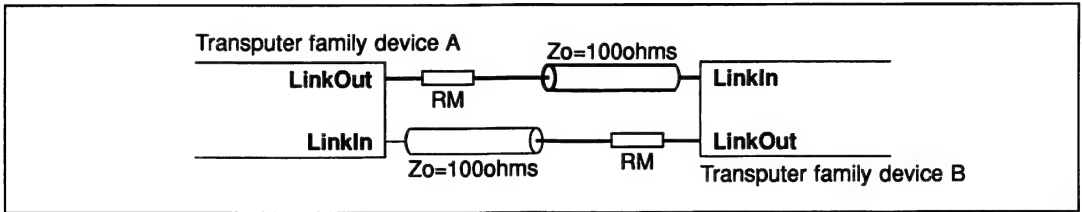


Figure 10.5 IMS T800 Links connected by transmission line

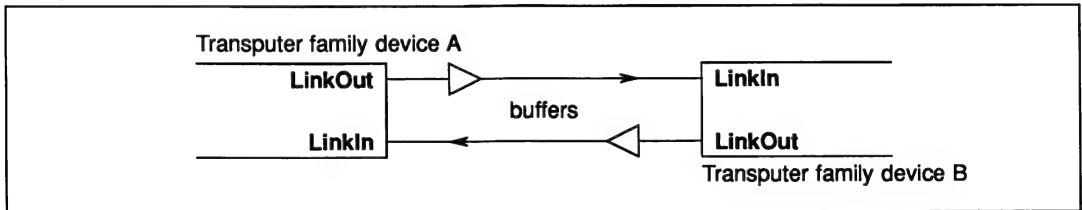


Figure 10.6 IMS T800 Links connected by buffers

## 11 Electrical specifications

### 11.1 DC electrical characteristics

Table 11.1 Absolute maximum ratings

SYMBOL	PARAMETER	MIN	MAX	UNITS	NOTE
VCC	DC supply voltage	0	7.0	V	1,2,3
VI, VO	Voltage on input and output pins	-0.5	VCC+0.5	V	1,2,3
II	Input current		±25	mA	4
OSCT	Output short circuit time (one pin)		1	s	2
TS	Storage temperature	-65	150	°C	2
TA	Ambient temperature under bias	-55	125	°C	2
PDmax	Maximum allowable dissipation		2	W	

#### Notes

- 1 All voltages are with respect to **GND**.
- 2 This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the operating sections of this specification is not implied. Stresses greater than those listed may cause permanent damage to the device. Exposure to absolute maximum rating conditions for extended periods may affect reliability.
- 3 This device contains circuitry to protect the inputs against damage caused by high static voltages or electrical fields. However, it is advised that normal precautions be taken to avoid application of any voltage higher than the absolute maximum rated voltages to this high impedance circuit. Unused inputs should be tied to an appropriate logic level such as **VCC** or **GND**.
- 4 The input current applies to any input or output pin and applies when the voltage on the pin is between **GND** and **VCC**.

Table 11.2 Operating conditions

SYMBOL	PARAMETER	MIN	MAX	UNITS	NOTE
VCC	DC supply voltage	4.75	5.25	V	1
VI, VO	Input or output voltage	0	VCC	V	1,2
CL	Load capacitance on any pin		60	pF	
TA	Operating temperature range IMS T800-S	0	70	°C	3
TA	Operating temperature range IMS T800-M	-55	125	°C	3

#### Notes

- 1 All voltages are with respect to **GND**.
- 2 Excursions beyond the supplies are permitted but not recommended; see DC characteristics.
- 3 Air flow rate 400 linear ft/min transverse air flow.

Table 11.3 DC characteristics

SYMBOL	PARAMETER	MIN	MAX	UNITS	NOTE
VIH	High level input voltage	2.0	VCC+0.5	V	1,2
VIL	Low level input voltage	-0.5	0.8	V	1,2
II	Input current @ GND<VI<VCC		±10	μA	1,2
VOH	Output high voltage @ IOH=2mA	VCC-1		V	1,2
VOL	Output low voltage @ IOL=4mA		0.4	V	1,2
IOS	Output short circuit current @ GND<VO<VCC	36	65	mA	1,2,3,7
		65	100	mA	1,2,4,7
		25	90	mA	1,2,3,8
		50	130	mA	1,2,4,8
IOZ	Tristate output current @ GND<VO<VCC		±10	μA	1,2
PD	Power dissipation		1.2	W	2,5
CIN	Input capacitance @ f=1MHz		7	pF	6
COZ	Output capacitance @ f=1MHz		10	pF	6

## Notes

- 1 All voltages are with respect to **GND**.
- 2 Parameters for IMS T800-S measured at 4.75V<VCC<5.25V and 0°C<TA<70°C.  
Parameters for IMS T800-M measured at 4.75V<VCC<5.25V and -55°C<TA<125°C.  
Input clock frequency = 5MHz.
- 3 Current sourced from non-link outputs.
- 4 Current sourced from link outputs.
- 5 Power dissipation varies with output loading and program execution.  
Power dissipation for processor operating at 20MHz.
- 6 This parameter is sampled and not 100% tested.
- 7 Parameter for IMS T800-S.
- 8 Parameter for IMS T800-M.

## 11.2 Equivalent circuits

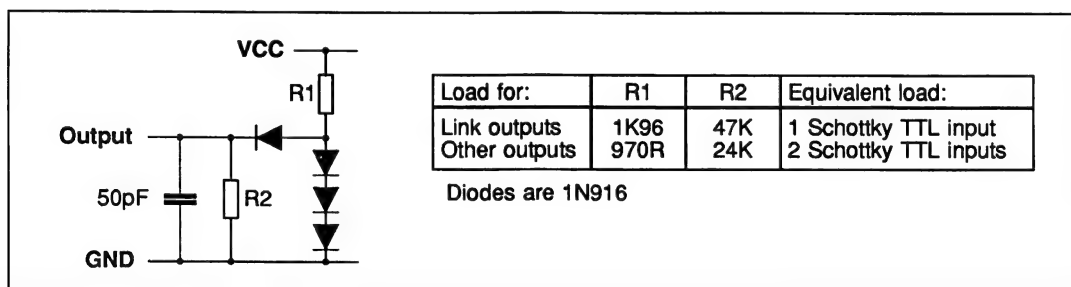


Figure 11.1 Load circuit for AC measurements

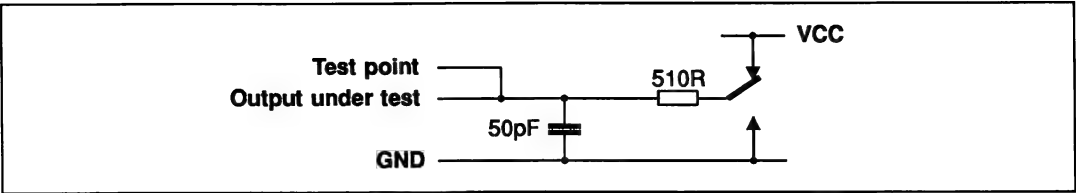


Figure 11.2 Tristate load circuit for AC measurements

11.3 AC timing characteristics

Table 11.4 Input, output edges

SYMBOL	PARAMETER	MIN	MAX	UNITS	NOTE
TDr	Input rising edges	2	20	ns	1,2
TDf	Input falling edges	2	20	ns	1,2
TQr	Output rising edges		25	ns	1
TQf	Output falling edges		15	ns	1
TS0LaHZ	Address high to tristate	a	a+6	ns	3
TS0LaLZ	Address low to tristate	a	a+6	ns	3

Notes

- 1 Non-link pins; see section on links.
- 2 All inputs except **ClockIn**; see section on **ClockIn**.
- 3 a is T2 where T2 can be from one to four periods Tm in length.  
Address lines include **MemnotWrD0**, **MemnotRfD1**, **MemAD2-31**.

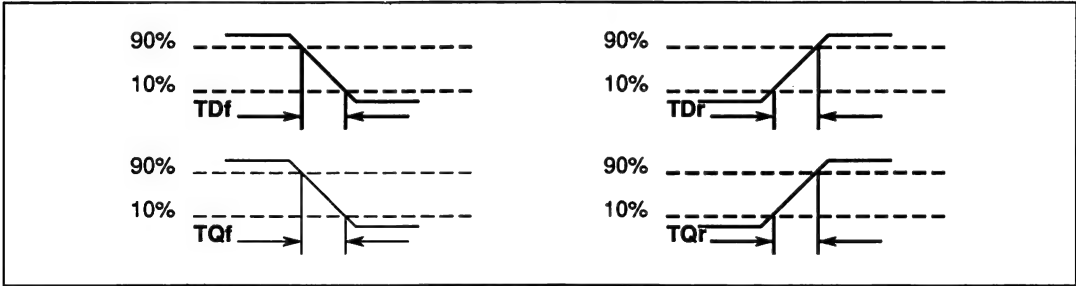


Figure 11.3 IMS T800 input and output edge timing

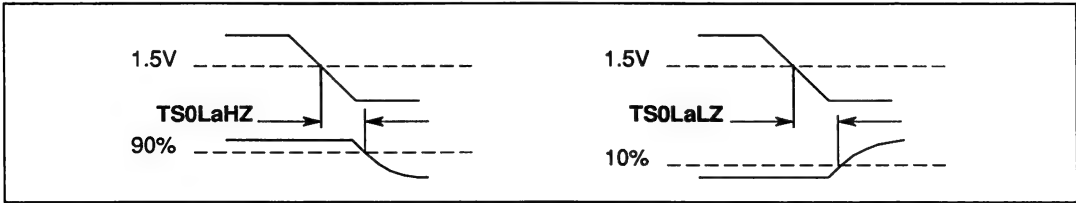


Figure 11.4 IMS T800 tristate timing relative to notMemS0

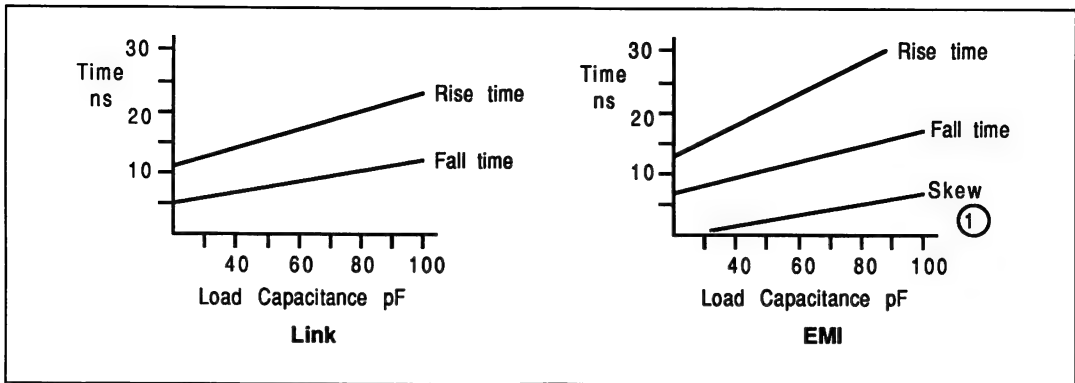


Figure 11.5 Typical rise/fall times

## Notes

- 1 Skew is measured between **notMemS0** with a standard load (2 Schottky TTL inputs and 30pF) and **notMemS0** with a load of 2 Schottky TTL inputs and varying capacitance.

## 11.4 Power rating

Internal power dissipation  $P_{INT}$  of transputer and peripheral chips depends on  $V_{CC}$ , as shown in figure 11.6.  $P_{INT}$  is substantially independent of temperature.

Total power dissipation  $P_D$  of the chip is

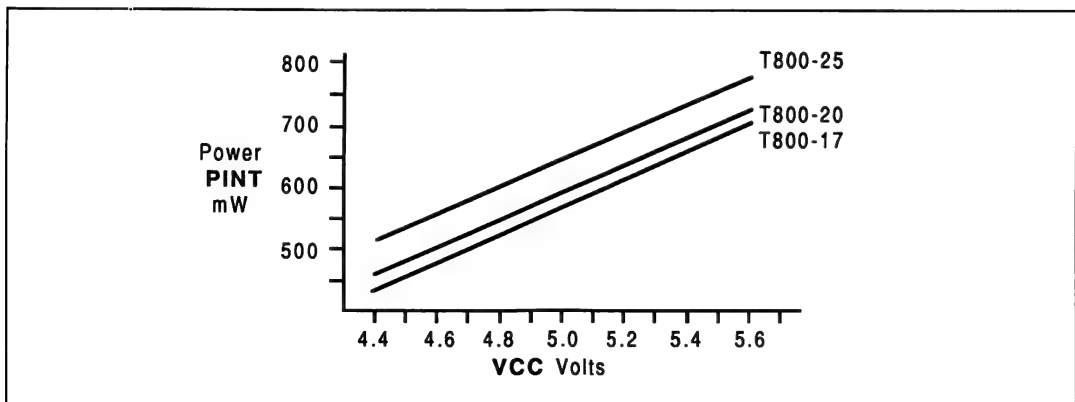
$$P_D = P_{INT} + P_{IO}$$

where  $P_{IO}$  is the power dissipation in the input and output pins; this is application dependent.

Internal working temperature  $T_J$  of the chip is

$$T_J = T_A + \theta_{JA} * P_D$$

where  $T_A$  is the external ambient temperature in °C and  $\theta_{JA}$  is the junction-to-ambient thermal resistance in °C/W.  $\theta_{JA}$  for each package is given in the Packaging Specifications section.

Figure 11.6 IMS T800 internal power dissipation vs  $V_{CC}$

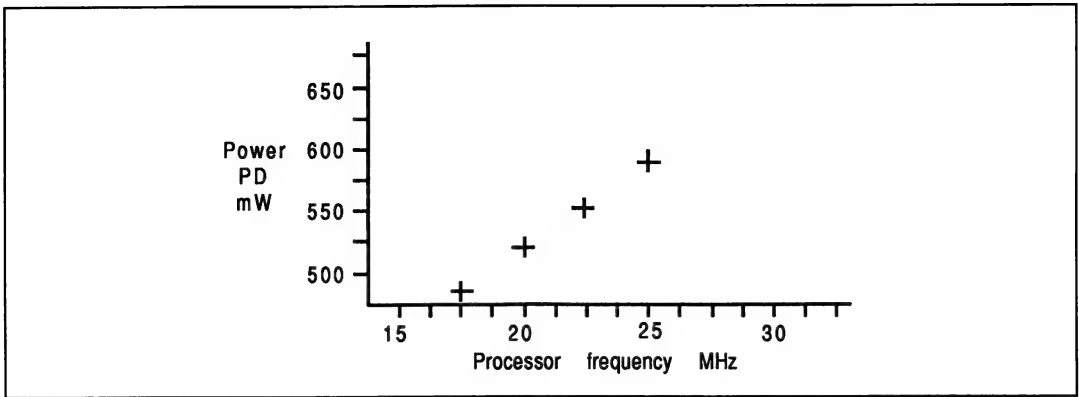


Figure 11.7 IMS T800 typical power dissipation with processor speed

## 12 Performance

The performance of the transputer is measured in terms of the number of bytes required for the program, and the number of (internal) processor cycles required to execute the program. The figures here relate to OCCAM programs. For the same function, other languages should achieve approximately the same performance as occam.

With transputers incorporating an FPU, this type of performance calculation is straight forward when considering only integer data types. However, when floating point calculations using the **REAL32** and **REAL64** data types are present in the program, complications arise due to the concurrency inherent in the transputer's design whereby integer calculations can be overlapped with floating point calculations. A more comprehensive guide to the impact of this concurrency on transputer performance can be found in The Transputer Instruction Set - A Compiler Writers' Guide.

### 12.1 Performance overview

These figures are averages obtained from detailed simulation, and should be used only as an initial guide; they assume operands are of type **INT**. The abbreviations in table 12.1 are used to represent the quantities indicated. In the replicator section of the table, figures in braces {} are not necessary if the number of replications is a compile time constant. To estimate performance, add together the time for the variable references and the time for the operation.

Table 12.1 Key to performance table

<b>np</b>	number of component processes
<b>ne</b>	number of processes earlier in queue
<b>r</b>	1 if <b>INT</b> parameter or array parameter, 0 if not
<b>ts</b>	number of table entries (table size)
<b>w</b>	width of constant in nibbles
<b>p</b>	number of places to shift
<b>Eg</b>	expression used in a guard
<b>Et</b>	timer expression used in a guard
<b>Tb</b>	most significant bit set of multiplier ((-1) if the multiplier is 0)
<b>Tbp</b>	most significant bit set in a positive multiplier when counting from zero ((-1) if the multiplier is 0)
<b>Tbc</b>	most significant bit set in the two's complement of a negative multiplier
<b>nsp</b>	Number of scalar parameters in a procedure
<b>nap</b>	Number of array parameters in a procedure

Table 12.2 Performance

	Size (bytes)	Time (cycles)
<b>Names</b>		
variables		
in expression	1.1+r	2.1+2(r)
assigned to or input to	1.1+r	1.1+(r)
in <b>PROC</b> or <b>FUNCTION</b> call,		
corresponding to an <b>INT</b> parameter	1.1+r	1.1+(r)
channels	1.1	2.1
<b>Array Variables</b> (for single dimension arrays)		
constant subscript	0	0
variable subscript	5.3	7.3
expression subscript	5.3	7.3
<b>Declarations</b>		
<b>CHAN OF protocol</b>	3.1	3.1
[ <b>size</b> ] <b>CHAN OF protocol</b>	9.4	2.2 + 20.2* <b>size</b>
<b>PROC</b>	body+2	0
<b>Primitives</b>		
assignment	0	0
input	4	26.5
output	1	26
<b>STOP</b>	2	25
<b>SKIP</b>	0	0
<b>Arithmetic operators</b>		
+    -	1	1
*	2	39
/	2	40
<b>REM</b>	2	38
>>   <<	2	3+p
<b>Modulo Arithmetic operators</b>		
<b>PLUS</b>	2	2
<b>MINUS</b>	1	1
<b>TIMES</b> (fast multiply, positive operand)	1	4+Tbp
<b>TIMES</b> (fast multiply, negative operand)	1	5+Tbc
<b>Boolean operators</b>		
<b>OR</b>	4	8
<b>AND</b> <b>NOT</b>	1	2
<b>Comparison operators</b>		
= constant	0	1
= variable	2	3
<> constant	1	3
<> variable	3	5
>    <	1	2
>=   <=	2	4
<b>Bit operators</b>		
^    v    <<    ~	2	2
<b>Expressions</b>		
constant in expression	w	w
check if error	4	6



Table 12.3 Performance

	Size (bytes)	Time (cycles)
<b>Timers</b>		
timer input	2	3
timer <b>AFTER</b>		
if past time	2	4
with empty timer queue	2	31
non-empty timer queue	2	$38+ne*9$
<b>ALT</b> (timer)		
with empty timer queue	6	52
non-empty timer queue	6	$59+ne*9$
timer alt guard	$8+2Eg+2Et$	$34+2Eg+2Et$
<b>Constructs</b>		
<b>SEQ</b>	0	0
<b>IF</b>	1.3	1.4
if guard	3	4.3
<b>ALT</b> (non timer)	6	26
alt channel guard	$10.2+2Eg$	$20+2Eg$
skip alt guard	$8+2Eg$	$10+2Eg$
<b>PAR</b>	$11.5+(np-1)*7.5$	$19.5+(np-1)*30.5$
<b>WHILE</b>	4	12
<b>Procedure or function call</b>		
	$3.5+(nsp-2)*1.1$ $+nap*2.3$	$16.5+(nsp-2)*1.1$ $+nap*2.3$
<b>Replicators</b>		
replicated <b>SEQ</b>	$7.3\{+5.1\}$	$(-3.8)+15.1*count\{+7.1\}$
replicated <b>IF</b>	$12.3\{+5.1\}$	$(-2.6)+19.4*count\{+7.1\}$
replicated <b>ALT</b>	$24.8\{+10.2\}$	$25.4+33.4*count\{+14.2\}$
replicated timer <b>ALT</b>	$24.8\{+10.2\}$	$62.4+33.4*count\{+14.2\}$
replicated <b>PAR</b>	$39.1\{+5.1\}$	$(-6.4)+70.9*count\{+7.1\}$

## 12.2 Fast multiply, **TIMES**

The IMS T800 has a fast integer multiplication instruction *product*. For a positive multiplier its execution time is  $4+Tbp$  cycles, and for a negative multiplier  $5+Tbc$  cycles (table 12.1). The time taken for a multiplication by zero is 3 cycles.

Implementations of high level languages on the transputer may take advantage of this instruction. For example, the OCCAM modulo arithmetic operator **TIMES** is implemented by the instruction and the right-hand operand is treated as the multiplier. The fast multiplication instruction is also used in high level language implementations for the multiplication implicit in multi-dimensional array access.

### 12.3 Arithmetic

A set of functions are provided within the development system to support the efficient implementation of multiple length integer arithmetic. In the IMS T800, floating point arithmetic is taken care of by the FPU. In table 12.4  $n$  gives the number of places shifted and all arguments and results are assumed to be local. Full details of these functions are provided in the OCCAM reference manual, supplied as part of the development system and available as a separate publication.

When calculating the execution time of the predefined maths functions, no time needs to be added for calling overhead. These functions are compiled directly into special purpose instructions which are designed to support the efficient implementation of multiple length integer arithmetic and floating point arithmetic.

Table 12.4 Arithmetic performance

Function	Cycles	+ cycles for parameter access †
<b>LONGADD</b>	2	7
<b>LONGSUM</b>	3	8
<b>LONGSUB</b>	2	7
<b>LONGDIFF</b>	3	8
<b>LONGPROD</b>	34	8
<b>LONGDIV</b>	36	8
<b>SHIFTRIGHT</b> ( $n < 32$ )	$4+n$	8
( $n \geq 32$ )	$n-27$	
<b>SHIFLEFT</b> ( $n < 32$ )	$4+n$	8
( $n \geq 32$ )	$n-27$	
<b>NORMALISE</b> ( $n < 32$ )	$n+6$	7
( $n \geq 32$ )	$n-25$	
( $n=64$ )	4	
<b>ASHIFTRIGHT</b>	<b>SHIFTRIGHT+2</b>	5
<b>ASHIFLEFT</b>	<b>SHIFLEFT+4</b>	5
<b>ROTATERIGHT</b>	<b>SHIFTRIGHT</b>	7
<b>ROTATELEFT</b>	<b>SHIFLEFT</b>	7
<b>FRACMUL</b>	<b>LONGPROD+4</b>	5

† Assuming local variables.

### 12.4 IMS T800 floating point operations

All references to **REAL32** or **REAL64** operands within programs compiled for the IMS T800 normally produce the following performance figures.

Table 12.5 Floating point performance

	Size (bytes)	REAL32 Time (cycles)	REAL64 Time (cycles)
<b>Names</b>			
variables			
in expression	3.1	3	5
assigned to or input to	3.1	3	5
in <b>PROC</b> or <b>FUNCTION</b> call,			
corresponding to a <b>REAL</b>			
parameter	1.1+r	1.1+r	1.1+r
<b>Arithmetic operators</b>			
+ -	2	7	7
*	2	11	20
/	2	17	32
<b>REM</b>	11	19	34
<b>Comparison operators</b>			
=	2	4	4
<>	3	6	6
> <	2	5	5
>= <=	3	7	7
<b>Conversions</b>			
<b>REAL32</b> to -	2		3
<b>REAL64</b> to -	2	6	
To <b>INT32</b> from -	5	9	9
To <b>INT64</b> from -	18	32	32
<b>INT32</b> to -	3	7	7
<b>INT64</b> to -	14	24	22

#### 12.4.1 IMS T800 floating point functions

These functions are provided by the development system. They are compiled directly into special purpose instructions designed to support the efficient implementation of some of the common mathematical functions of other languages. The functions provide **ABS** and **SQRT** for both **REAL32** and **REAL64** operand types.

Table 12.6 IMS T800 floating point arithmetic performance

Function	Cycles	+ cycles for parameter access †	
		REAL32	REAL64
<b>ABS</b>	2	8	
<b>SQRT</b>	118	8	
<b>DABS</b>	2		12
<b>DSQRT</b>	244		12

† Assuming local variables.

### 12.4.2 IMS T800 special purpose functions and procedures

The functions and procedures given in tables 12.8 and 12.9 are provided by the development system to give access to the special instructions available on the IMS T800. Table 12.7 shows the key to the table.

Table 12.7 Key to special performance table

<b>Tb</b>	most significant bit set in the word counting from zero
<b>n</b>	number of words per row (consecutive memory locations)
<b>r</b>	number of rows in the two dimensional move
<b>nr</b>	number of bits to reverse

Table 12.8 Special purpose functions performance

Function	Cycles	+ cycles for parameter access †
<b>BITCOUNT</b>	$2+Tb$	2
<b>CRCBYTE</b>	11	8
<b>CRCWORD</b>	35	8
<b>BITREVNBIT</b>	$5+nr$	4
<b>BITREVWORD</b>	36	2

† Assuming local variables.

Table 12.9 Special purpose procedures performance

Procedure	Cycles	+ cycles for parameter access †
<b>MOVE2D</b>	$8+(2n+23)*r$	8
<b>DRAW2D</b>	$8+(2n+23)*r$	8
<b>CLIP2D</b>	$8+(2n+23)*r$	8

† Assuming local variables.

### 12.5 Effect of external memory

Extra processor cycles may be needed when program and/or data are held in external memory, depending both on the operation being performed, and on the speed of the external memory. After a processor cycle which initiates a write to memory, the processor continues execution at full speed until at least the next memory access.

Whilst a reasonable estimate may be made of the effect of external memory, the actual performance will depend upon the exact nature of the given sequence of operations.

External memory is characterized by the number of extra processor cycles per external memory cycle, denoted as **e**. For the IMS T800, with the fastest external memory the value of **e** is 2; a typical value for a large external memory is 5.

If a program is stored in external memory, and **e** has the value 2 or 3, then no extra cycles need be estimated for linear code sequences. For larger values of **e**, the number of extra cycles required for linear code sequences may be estimated at  $(e-3)/4$ . A transfer of control may be estimated as requiring  $e+3$  cycles.

These estimates may be refined for various constructs. In table 12.10 **n** denotes the number of components in a construct. In the case of **IF**, the **n**'th conditional is the first to evaluate to **TRUE**, and the costs include the

costs of the conditionals tested. The number of bytes in an array assignment or communication is denoted by  $b$ .

Table 12.10 External memory performance

	IMS T800	
	Program off chip	Data off chip
Boolean expressions	$e-2$	0
<b>IF</b>	$3en-8$	en
Replicated <b>IF</b>	$(6e-4)n+7$	$(5e-2)n+8$
Replicated <b>SEQ</b>	$(3e-3)n+2$	$(4e-2)n$
<b>PAR</b>	$(3e-1)n+8$	$3en+4$
Replicated <b>PAR</b>	$(10e-8)n+8$	$16en-12$
<b>ALT</b>	$(2e-4)n+6e$	$(2e-2)n+10e-8$
Array assignment and communication in one transputer	0	$\max(2e, e(b/2))$

The following simulation results illustrate the effect of storing program and/or data in external memory. The results are normalized to 1 for both program and data on chip. The first program (Sieve of Erastosthenes) is an extreme case as it is dominated by small, data access intensive loops; it contains no concurrency, communication, or even multiplication or division. The second program is the pipeline algorithm for Newton Raphson square root computation.

Table 12.11 IMS T800 external memory performance

	Program	e=2	e=3	e=4	e=5	On chip
<b>Program off chip</b>	1	1.3	1.5	1.7	1.9	1
	2	1.1	1.2	1.2	1.3	1
<b>Data off chip</b>	1	1.5	1.8	2.1	2.3	1
	2	1.2	1.4	1.6	1.7	1
<b>Program and data off chip</b>	1	1.8	2.2	2.7	3.2	1
	2	1.3	1.6	1.8	2.0	1

## 12.6 Interrupt latency

If the process is a high priority one and no other high priority process is running, the latency is as described in table 12.12. The timings given are in full processor cycles **TPCLPCL**; the number of  $T_m$  states is also given where relevant. Maximum latency assumes all memory accesses are internal ones.

Table 12.12 Interrupt latency

	Typical		Maximum	
	TPCLPCL	$T_m$	TPCLPCL	$T_m$
IMS T800 with FPU in use	19	38	78	156
IMS T800 with FPU not in use	19	38	58	116

### 13 Package specifications

#### 13.1 84 pin grid array package

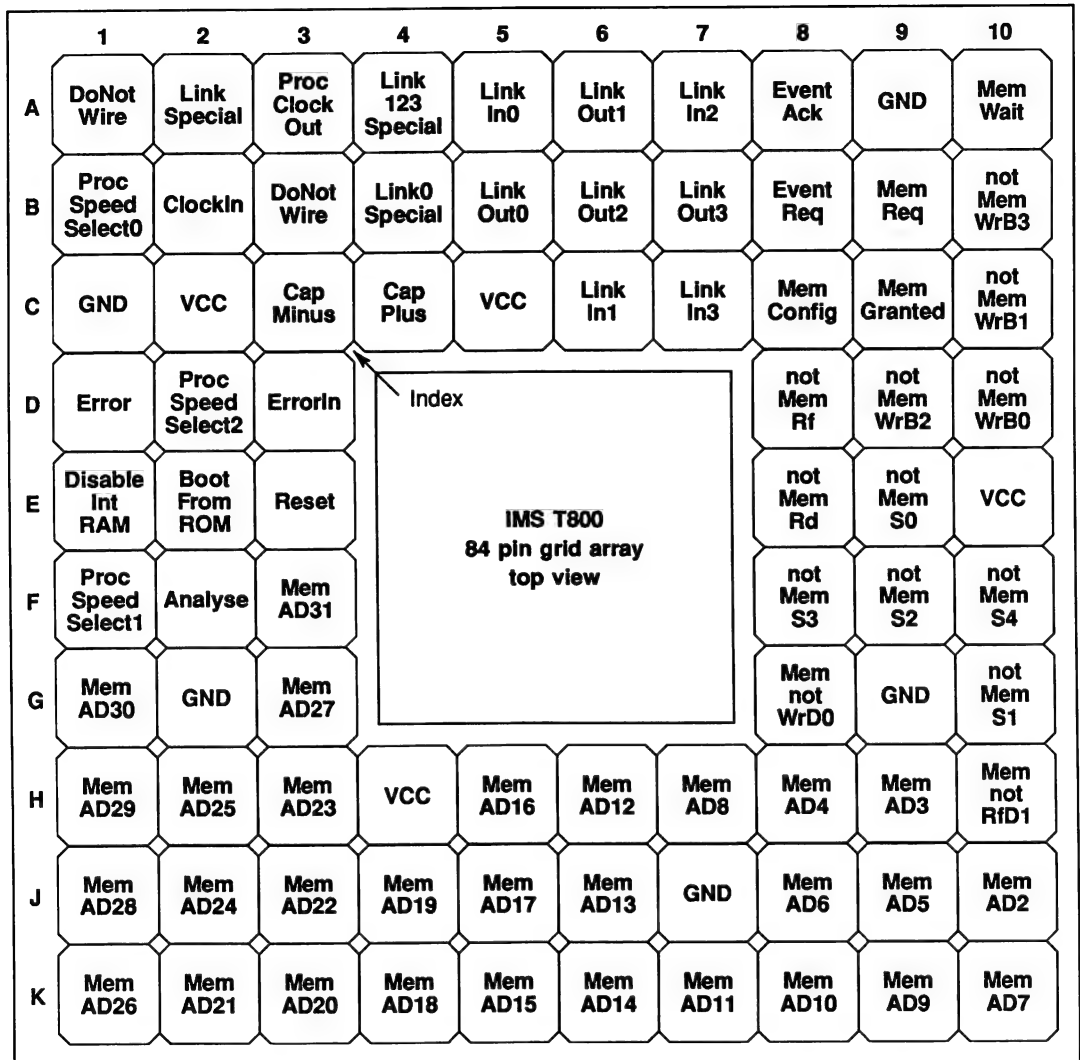


Figure 13.1 IMS T800 84 pin grid array package pinout

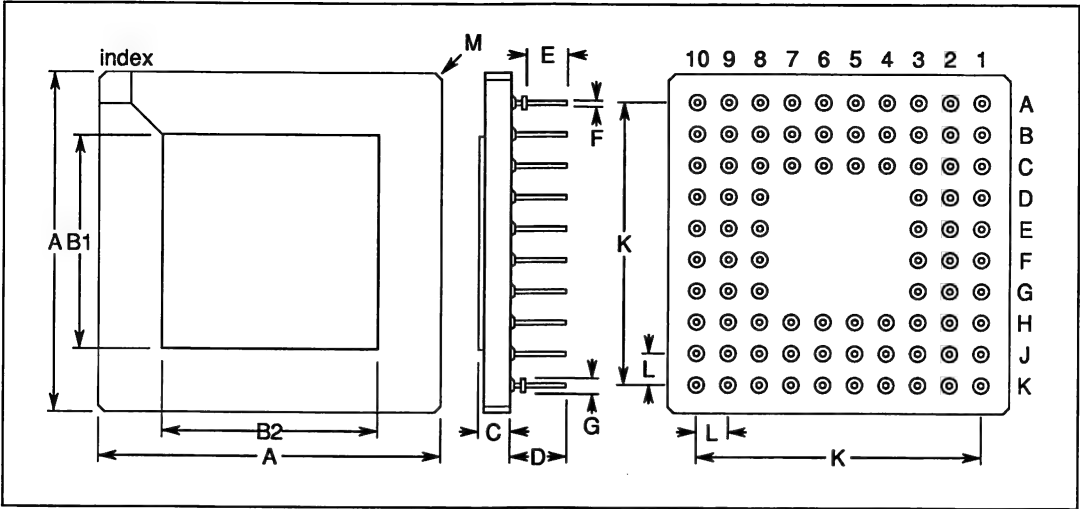


Figure 13.2 84 pin grid array package dimensions

Table 13.1 84 pin grid array package dimensions

DIM	Millimetres		Inches		Notes
	NOM	TOL	NOM	TOL	
A	26.924	±0.254	1.060	±0.010	Pin diameter Flange diameter
B1	17.019	±0.127	0.670	±0.005	
B2	18.796	±0.127	0.740	±0.005	
C	2.456	±0.278	0.097	±0.011	
D	4.572	±0.127	0.180	±0.005	
E	3.302	±0.127	0.130	±0.005	
F	0.457	±0.025	0.018	±0.002	
G	1.143	±0.127	0.045	±0.005	
K	22.860	±0.127	0.900	±0.005	
L	2.540	±0.127	0.100	±0.005	
M	0.508		0.020		Chamfer

Package weight is approximately 7.2 grams

Table 13.2 84 pin grid array package junction to ambient thermal resistance

SYMBOL	PARAMETER	MIN	NOM	MAX	UNITS	NOTE
$\theta_{JA}$	At 400 linear ft/min transverse air flow			35	°C/W	

## 14 Ordering

This section indicates the designation of speed and package selections for the various devices. Speed of **ClockIn** is 5 MHz for all parts. Transputer processor cycle time is nominal; it can be calculated more exactly using the phase lock loop factor **PLLx**, as detailed in the external memory section.

For availability contact local INMOS sales office or authorised distributor.

Table 14.1 IMS T800 ordering details

INMOS designation	Processor clock speed	Processor cycle time	PLLx	Package
IMS T800-G17S	17.5 MHz	57 ns	3.5	Ceramic Pin Grid
IMS T800-G20S	20.0 MHz	50 ns	4.0	Ceramic Pin Grid
IMS T800-G25S	25.0 MHz	40 ns	5.0	Ceramic Pin Grid
IMS T800-G30S	30.0 MHz	33 ns	6.0	Ceramic Pin Grid
IMS T800-G17M	17.5 MHz	57 ns	3.5	Ceramic Pin Grid MIL Spec
IMS T800-G20M	20.0 MHz	50 ns	4.0	Ceramic Pin Grid MIL Spec

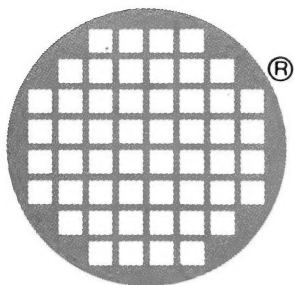
The timing parameters in this datasheet are based on full characterisation of the 17 MHz and 20 MHz parts. Data for higher speeds is based on tests on a limited number of samples and may change when full characterisation is completed.







17



**inmos®**

**INMOS Limited** • 1000 Aztec West • Almondsbury • Bristol BS12 4SQ • U.K. • Phone (0454) 616616 • TLX 444723

**INMOS Corporation** • P.O. Box 16000 • Colorado Springs • Colorado 80935 • USA • Phone (303) 630 4000 • TWX 910 920 4904


**INMOS GmbH** • Danziger Strasse 2 • 8057 Eching • West Germany • Phone (089) 319 10 28 • TLX 522645

**INMOS Japan K.K.** • 4th Floor No 1 Kowa Bldg • 11-41 Akasaka 1-chome • Minato-ku • Tokyo 107 • Japan • Phone 03-505-2840 •

FAX 03-505-2844 • TLX J29507 TEI JPN

**INMOS SARL** • Immeuble Monaco • 7 rue Le Corbusier • SILIC 219 • 94518 Rungis Cedex • France • Phone (1) 46.87.22.01 • TLX 201222

INMOS reserves the right to make changes in specifications at any time and without notice. The information furnished by INMOS in this publication is believed to be accurate; however, no responsibility is assumed for its use, nor for any infringement of patents or other rights of third parties resulting from its use. No licence is granted under any patents, trade marks or other rights of INMOS.

 **inmos**, **IMS** and **occam** are trade marks of the INMOS Group of Companies.